查询mega48/mega88/mega168供应商

产品特性

- 高性能、低功耗的 8 位 AVR® 微处理器
- 先进的 RISC 结构
  - 131 条指令 – 大多数指令的执行时间为单个时钟周期
  - 32 x 8 通用工作寄存器
  - 全静态操作
  - 工作于 16 MHz 时性能高达 16 MIPS
  - 只需两个时钟周期的硬件乘法器
- 非易失性的程序和数据存储器
  - 4/8/16K 字节的系统内可编程 Flash (ATmega48/88/168)
    擦写寿命 : 10,000 次
  - 具有独立锁定位的可选 Boot 代码区
    通过片上 Boot 程序实现系统内编程
    真正的同时读写操作
  - 256/512/512 字节的 EEPROM (ATmega48/88/168)
    擦写寿命 : 100,000 次
  - 512/1K/1K 字节的片内 SRAM (ATmega48/88/168)
  - 可以对锁定位进行编程以实现用户程序的加密
- 外设特点
  - 两个具有独立预分频器和比较器功能的 8 位定时器 / 计数器
  - 一个具有预分频器、比较功能和捕捉功能的 16 位定时器 / 计数器
  - 具有独立振荡器的实时计数器 RTC
  - 六通道 PWM
  - 8 路 10 位 ADC(TQFP 与 MLF 封装)
  - 6 路 10 位 ADC(PDIP 封装)
  - 可编程的串行 USART 接口
  - 可工作于主机 / 从机模式的 SPI 串行接口
  - 面向字节的两线串行接口
  - 具有独立片内振荡器的可编程看门狗定时器
  - 片内模拟比较器
  - 引脚电平变化可引发中断及唤醒 MCU
- 特殊的微控制器特点
  - 上电复位以及可编程的掉电检测
  - 经过标定的片内 RC 振荡器
  - 片内 / 外中断源
  - 五种休眠模式：空闲模式、 ADC 噪声抑制模式、省电模式、掉电模式和 Standby 模式
- I/O 口与封装
  - 23 个可编程的 I/O 口线
  - 32 引脚 TQFP 封装 与 32 引脚 MLF 封装
- 工作电压 :
  - ATmega48V/88V/168V：1.8 - 5.5V
  - ATmega48/88/168：2.7 - 5.5V
- 工作温度范围 :
  - -40°C 至 85°C
- 工作速度等级 :
  - ATmega48V/88V/168V: 0 - 2 MHz @ 1.8 - 5.5V, 0 - 8 MHz @ 2.4 - 5.5V
  - ATmega48/88/168: 0 - 8 MHz @ 2.7 - 5.5V, 0 - 16 MHz @ 4.5 - 5.5V
- 极低功耗
  - 正常模式 :
    1 MHz, 1.8V: 300µA
    32 kHz, 1.8V: 20µA (包括振荡器)
  - 掉电模式 :
    1.8V， 0.5µA

具有 8K 系统内可编程 Flash 的 8 位 AVR® 微控制器

ATmega48/V
ATmega88/V
ATmega168/V

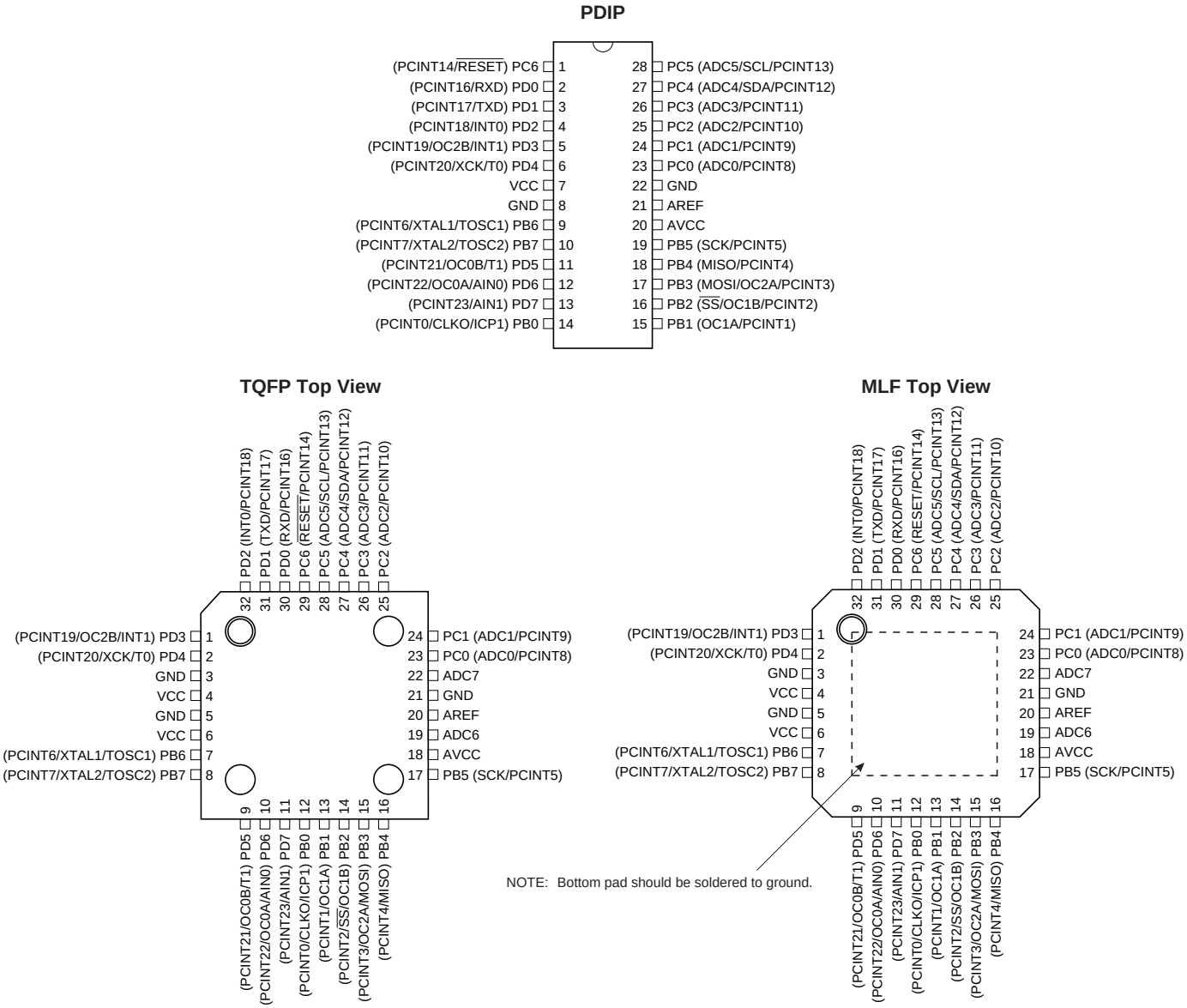
初稿

本文是英文数据手册的中文翻译，其目的是方便中国用户的阅读。它无法自动跟随原稿的更新，同时也可能存在翻译上的错误。读者应该以英文原稿为参考以获得更准确的信息。

Rev. 2545A–AVR–09/03

引脚配置

Figure 1. ATmega48/88/168 引脚排列

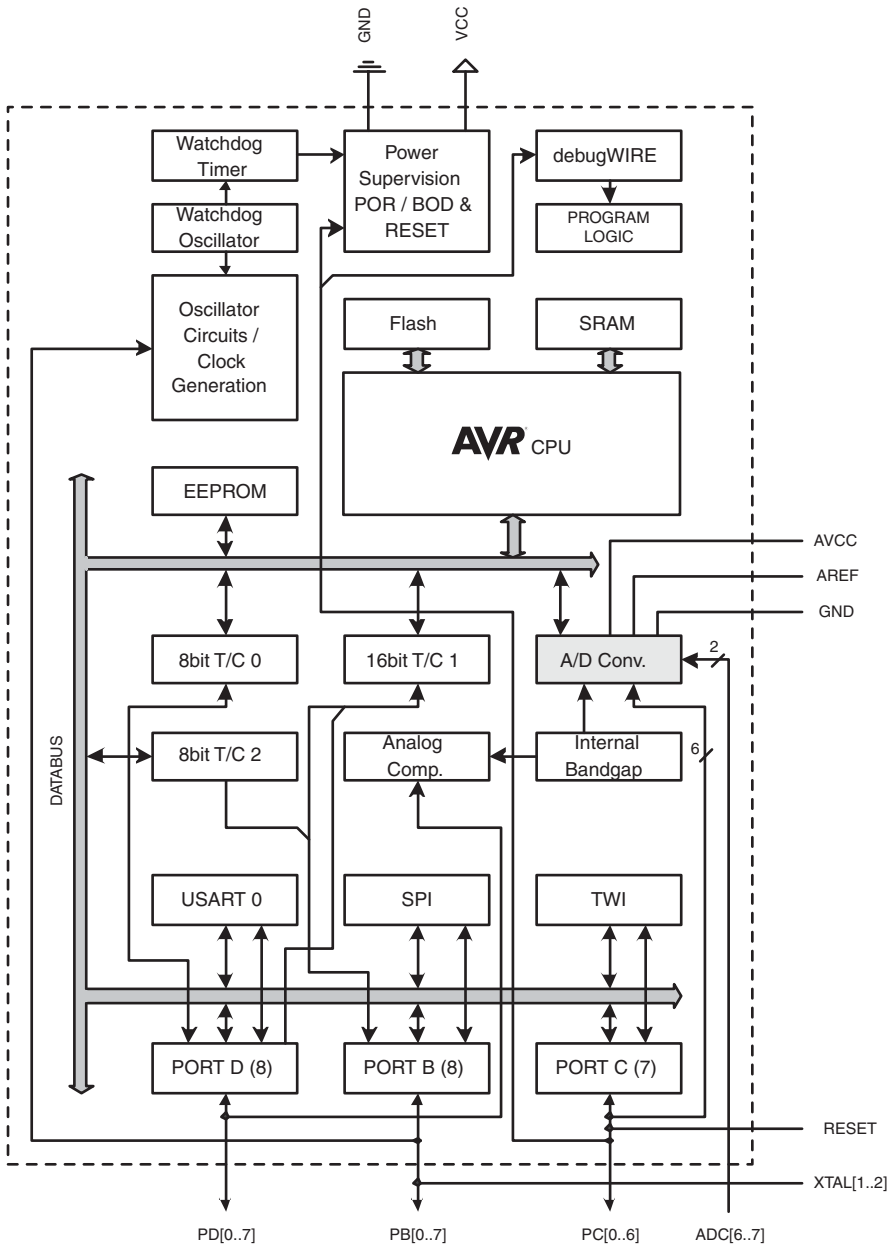


综述

ATmega48/88/168是基于AVR增强型RISC结构的低功耗8位CMOS微控制器。由于其先进的指令集以及单时钟周期指令执行时间，ATmega48/88/168 的数据吞吐率高达1 MIPS/MHz，从而可以缓减系统在功耗和处理速度之间的矛盾。

方框图

Figure 2. 结构框图



AVR 内核具有丰富的指令集和 32 个通用工作寄存器。所有的寄存器都直接与算术逻辑单元 (ALU) 相连接，使得一条指令可以在一个时钟周期内同时访问两个独立的寄存器。这种结构大大提高了代码效率，并且具有比普通的 CISC 微控制器最高至 10 倍的数据吞吐率。

ATmega48/88/168有如下特点: 4K/8K/16K字节的系统内可编程Flash(具有在编程过程中还可以读的能力，即 RWW)， 256/512/512 字节 EEPROM， 512/1K/1K 字节 SRAM，23 个通用 I/O 口线， 32 个通用工作寄存器，三个具有比较模式的灵活的定时器 / 计数器 (T/C), 片内 / 外中断，可编程串行 USART，面向字节的两线串行接口，一个 SPI 串行端口，一个 6 路 10 位 ADC (TQFP 与 MLF 封装的器件具有 8 路 10 位 ADC)，具有片内振荡器的可编程看门狗定时器，以及五种可以通过软件选择的省电模式。空闲模式时 CPU 停止工作，而 SRAM、 T/C、 USART、两线串行接口、 SPI 端口以及中断系统继续工作；掉电模式时晶体振荡器停止振荡，所有功能除了中断和硬件复位之外都停止工作，寄存器的内容则一直保持；省电模式时异步定时器继续运行，以允许用户维持时间基准，器件的其他部分则处于睡眠状态； ADC 噪声抑制模式时 CPU 和所有的 I/O 模块停止运行，而异步定时器和 ADC 继续工作，以减少 ADC 转换时的开关噪声； Standby 模式时振荡器工作而其他部分睡眠，使得器件只消耗极少的电流，同时具有快速启动能力。

ATmega48/88/168 是以 Atmel 的高密度非易失性内存技术生产的。片内 ISP Flash 可以通过 SPI 接口、通用编程器，或引导程序进行多次编程。引导程序可以使用任意接口将应用程序来下载到应用 Flash 存储区。在更新应用 Flash 存储区时引导程序区的代码继续运行，从而实现了 FLASH 的 RWW 操作。通过将 8 位 RISC CPU 与系统内可编程的 Flash 集成在一个芯片内， ATmega48/88/168 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATmega48/88/168 AVR 有整套的开发工具，包括 C 编译器，宏汇编，程序调试器 / 仿真器和评估板。

ATmega48,ATmega88, 与 ATmega168 的兼容性

ATmega48、ATmega88 与 ATmega168 只是在存储器大小、boot loader 支持及中断向量长度上存在差别。Table1 给出了三种器件在存储器与中断向量长度方面的差别。

Table 1. 存储器大小简述

器件	Flash	EEPROM	RAM	中断向量长度
ATmega48	4K 字节	256 字节	512 字节	一个指令字 (16 位)
ATmega88	8K 字节	512 字节	1K 字节	一个指令字 (16 位)
ATmega168	16K 字节	512 字节	1K 字节	两个指令字 (32 位)

ATmega88 与 ATmega168 支持真正的同时读写自编程操作。芯片具有独立的 Boot Loader 区，SPM 指令只能在这个 FLASH 区里得到执行。而 ATmega48 不支持同时读写操作，它没有独立的 Boot Loader 区， SPM 指令可以访问整个 Flash 区。

声明

本数据手册中的典型值来源于对器件的仿真，以及其他基于相同产生工艺的 AVR 微控制器的标定特性。本器件经过特性化之后将给出实际的最大值和最小值。

引脚说明

VCC 数字电路的电源

GND 地

端口 B (PB7..0) XTAL1/ XTAL2/TOSC1/TOSC2 端口 B 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 B 保持为高阻态。

通过对系统时钟选择位的设定，PB6可作为反向振荡放大器与内部时钟操作电路的输入。

通过对系统时钟选择位的设定， PB7 可作为反向振荡放大器的输出。

系统使用内部 RC 振荡器时，通过设置 ASSR 寄存器的 AS2 位，可以将 PB7..6 作为异步定时器 / 计数器 2 的输入口 TOSC2..1 使用。

端口 B 也可以用做其他不同的特殊功能，请参见 P66” 端口 B 的第二功能 ” 与 P23” 系统时钟及其选项 ”。

端口 C (PC5..0)

端口 C 为 7 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 C 保持为高阻态。

PC6/$\overline{\text{RESET}}$

RSTDISBL位被编程时，可将PC6作为一个I/O口使用。因此，PC6引脚与端口C其他引脚的电特性是有区别的。

RSTDISBL位未编程时，PC6将作为复位输入引脚Reset。此时，即使系统时钟没有运行，该引脚上出现的持续时间超过最小脉冲宽度的低电平将产生复位信号。最小脉冲宽度在P37Table20 中给出。持续时间不到最小脉冲宽度的低电平不会产生复位信号。

端口 C 也可以用做其他不同的特殊功能，请参见 P69” 端口 C 的第二功能 ”。

端口 D (PD7..0)

端口 D 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 D 呈现为三态。

端口 D 也可以用做其他不同的特殊功能，请参见 P72” 端口 D 的第二功能 ”。

AVCC

AVCC 为 A/D 转换器的电源。当引脚 PC3..0 与 PC7..6 用于 ADC 时，AVCC 应通过一个低通滤波器与 V_{CC} 连接。不使用 ADC 时该引脚应直接与 V_{CC} 连接。 PC6..4 的电源则是由 V_{CC} 提供的。

AREF

AREF 为 ADC 的模拟基准输入引脚。

ADC7..6 (TQFP 与 MLF 封装)

TQFP 与 MLF 封装芯片的 ADC7..6 引脚为两个 10 位 A/D 转换器的输入口，它们的电压由 AVCC 提供。

代码例子

本手册包含了一些简单的代码例子以说明如何使用芯片各个不同的功能。这些例子都假定在编译之前已经包含了正确的头文件。有些 C 编译器在头文件里并没有包含位定义和中断，而且各个 C 编译器对中断处理有自己不同的处理方式。请注意查阅其文档以获取具体的信息。

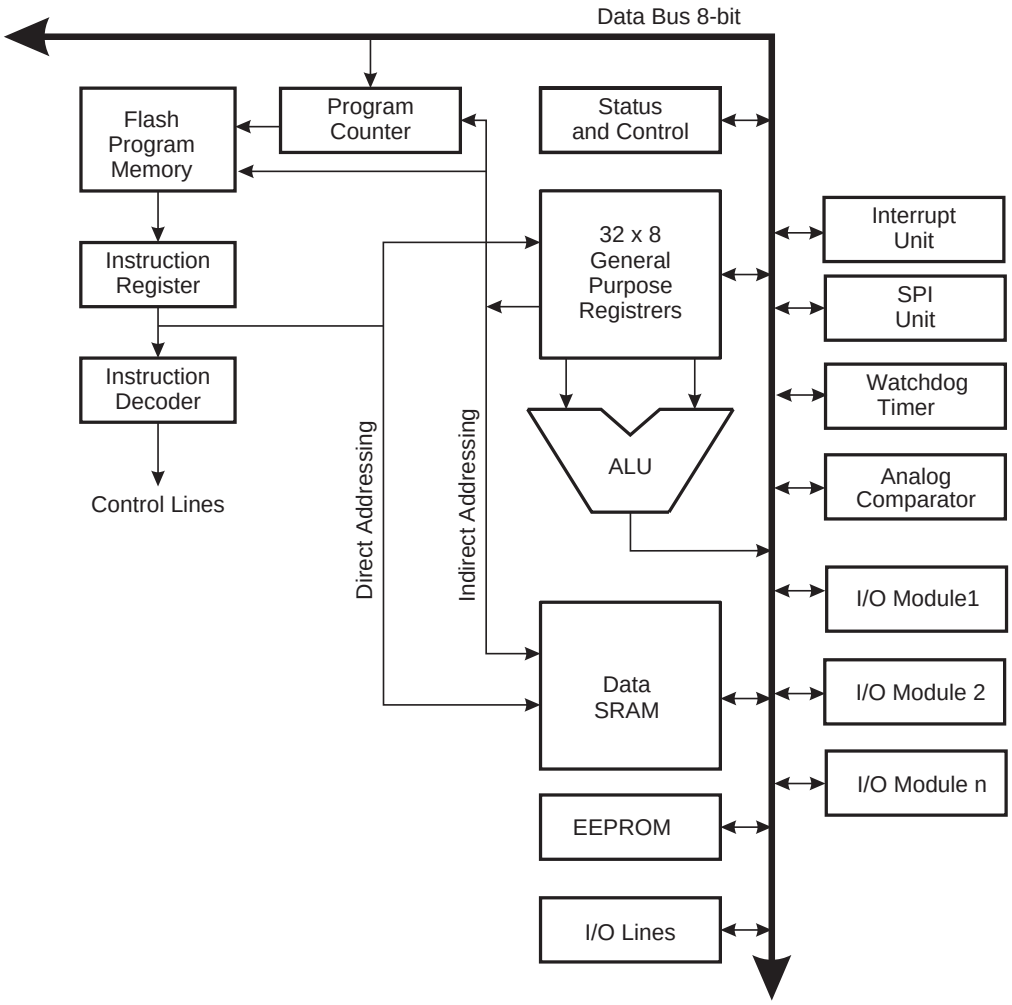
AVR CPU 内核

介绍

本节从总体上讨论 AVR 内核的结构。CPU 的主要任务是保证程序的正确执行。因此它必须能够访问存储器，执行运算，控制外设以及处理中断。

结构综述

Figure 3. AVR 结构的方框图



为了得到最大程度的性能以及并行性， AVR 采用了 Harvard 结构，具有独立的数据和程序总线。程序存储器的指令通过一级流水线运行。 CPU 在执行一条指令的同时读取下一条指令（在本文称为预取）。这个概念实现了指令的单时钟周期运行。程序存储器为可以在线编程的 FLASH。

快速访问寄存器文件包括 32 个 8 位通用工作寄存器，访问时间为一个时钟周期。从而可以实现单时钟周期的 ALU 操作。在典型的 ALU 操作过程中，两个位于寄存器文件的操作数同时被访问，然后执行相应的运算，结果再送回寄存器文件。整个过程仅需要一个时钟周期。

寄存器文件里有 6 个寄存器可以用作 3 个 16 位的间接寻址寄存器指针以寻址数据空间，实现高效的地址运算。其中一个指针还可以作为程序存储器查询表的地址指针。这些附加的功能寄存器即为 16 位的 X、 Y、 Z 寄存器。

ALU支持寄存器之间以及寄存器和常数之间的算术和逻辑运算。ALU也可以执行单寄存器操作。运算完成之后状态寄存器的内容将更新以反映操作结果。

程序流程通过有 / 无条件的跳转指令和调用指令来控制，从而直接寻址整个地址空间。大多数指令长度为 16 位，亦即每个程序存储器地址都包含一条 16 位或 32 位的指令。

程序存储器空间分为两个区：引导程序区和应用程序区。这两个区都有专门的锁定位以实现读和读 / 写保护。写应用程序区的 SPM 指令必须位于引导程序区。

在中断和调用子程序时返回地址的程序计数器 (PC) 保存于堆栈之中。堆栈位于通用数据 SRAM，故此嵌套深度仅受限于 SRAM 的大小。在复位例程里用户首先要初始化堆栈指针 SP。这个指针位于 I/O 空间，可以进行读写访问。数据 SRAM 可以通过 5 种不同的寻址模式进行访问。

AVR 存储器为线性的平面结构。

AVR 具有一个灵活的中断模块。控制寄存器位于 I/O 空间。状态寄存器里有全局中断使能位。每个中断在中断向量表里都有独立的中断向量。各个中断的优先级与其在中断向量表的位置有关，中断向量地址越低，优先级越高。

I/O 存储器空间包含 64 个可以直接寻址的地址，作为 CPU 外设的控制寄存器、SPI，以及其他 I/O 功能。映射到数据空间即为寄存器文件之后的地址 0x20 - 0x5F。此外，ATmega48/88/168 还有位于 SRAM 地址 0x60 - 0xFF 的扩展 I/O 空间，这些地址只能使用 ST/STS/STD 和 LD/LDS/LDD 指令来访问。

ALU—算术逻辑单元

AVR ALU 与 32 个通用工作寄存器直接相连。寄存器与寄存器之间、寄存器与立即数之间的 ALU 运算只需要一个时钟周期。ALU 操作分为 3 类：算术、逻辑和位操作。此外还提供了支持无 / 有符号数和分数乘法的乘法器。具体请参见“指令集”。

状态寄存器

状态寄存器包含了最近执行的算术指令的结果信息。这些信息可以用来改变程序流程以实现条件操作。如指令集所述，所有 ALU 运算都将影响状态寄存器的内容。这样，在许多情况下就不需要专门的比较指令了，从而使系统运行更快速，代码效率更高。

在进入中断服务程序时状态寄存器不会自动保存；中断返回时也不会自动恢复。这些工作需要软件来处理。

AVR 中断寄存器 SREG 定义如下：

位	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – I: 全局中断使能**

置位时使能全局中断。单独的中断使能由其他独立的控制寄存器控制。如果 I 清零，则不论单独中断标志置位与否，都不会产生中断。任意一个中断发生后 I 清零，而执行 RETI 指令后置位以使能中断。 I 也可以通过 SEI 和 CLI 指令来置位和清零。

- **位 6 – T: 位拷贝存储**

位拷贝指令 BLD 和 BST 利用 T 作为目的或源地址。BST 把寄存器的某一位拷贝到 T，而 BLD 把 T 拷贝到寄存器的某一位。

- **位 5 – H: 半进位标志**

半进位标志 H 表示算术操作发生了半进位。此标志对于 BCD 运算非常有用。详见 “ 指令集 ” 的说明。

- **位 4 – S: 符号位， S = N ⊕ V**

S 为负数标志 N 与 2 的补码溢出标志 V 的异或。详见指令集的说明。

- **位 3 – V: 2 的补码溢出标志**

支持 2 的补码运算。详见指令集的说明。

- **位 2 – N: 负数标志**

表明算术或逻辑操作结果为负。详见指令集的说明。

- **位 1 – Z: 零标志**

表明算术或逻辑操作结果为零。详见指令集的说明。

- **位 0 – C: 进位标志**

表明算术或逻辑操作发生了进位。详见 指令集的说明。

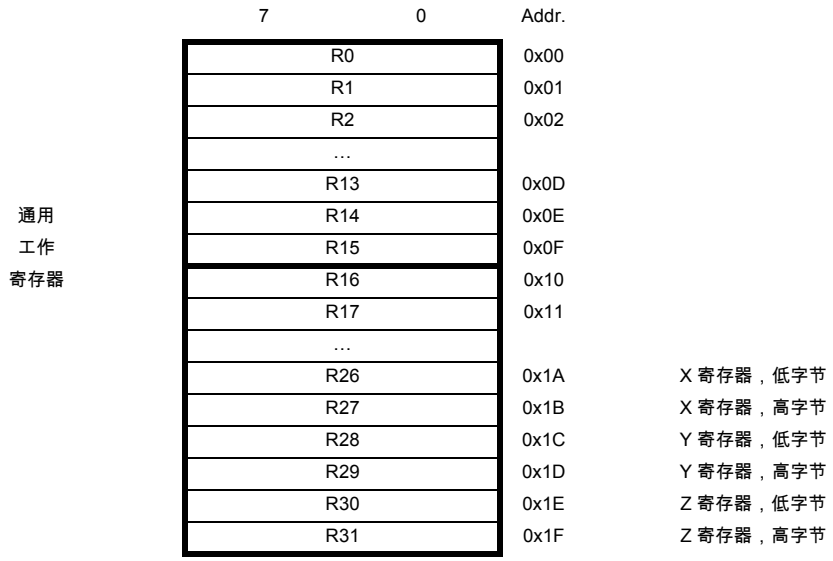
通用寄存器文件

寄存器文件针对 AVR 增强型 RISC 指令集做了优化。为了获得需要的性能和灵活性，寄存器文件支持以下的输入 / 输出方案：

- 输出一个 8 位操作数，输入一个 8 位结果
- 输出两个 8 位位操作数，输入一个 8 位结果
- 输出两个 8 位位操作数，输入一个 16 位结果
- 输出一个 16 位位操作数，输入一个 16 位结果

Figure 4 为 CPU 32 个通用工作寄存器的结构。

Figure 4. AVR CPU 通用工作寄存器



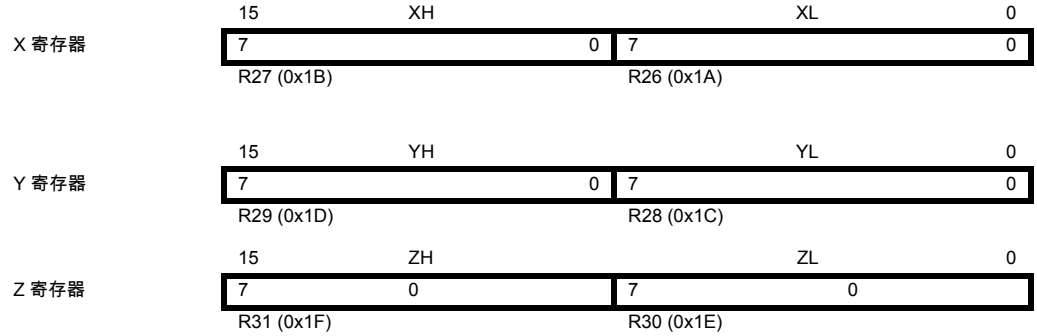
大多数操作寄存器文件的指令都可以直接访问所有的寄存器，而且多数这样的指令的执行时间为单时钟周期。

如 Figure 4 所示，每个寄存器都有一个数据内存地址将他们直接映射到用户数据空间的头32个地址。虽然寄存器文件的物理实现不是SRAM，这种内存组织方式在访问寄存器方面具有极大的灵活性，因为 X、 Y、 Z 寄存器可以设置为指向任意寄存器的指针。

X、Y、Z 寄存器

寄存器 R26..R31 除了用作通用寄存器外，还可以作为数据间接寻址用的地址指针。这三个间接寻址寄存器示于 Figure 5。

Figure 5. X、 Y、 Z 寄存器



在不同的寻址模式中，这些地址寄存器可以实现固定偏移量，自动加一和自动减一操作。具体细节请参见指令集。

堆栈指针

堆栈指针主要用来保存临时数据、局部变量和中断 / 子程序的返回地址。堆栈指针总是指向堆栈的顶部。要注意 AVR 的堆栈是向下生长的，即新数据推入堆栈时，堆栈指针的数值将减小。

堆栈指针指向数据 SRAM 堆栈区。在此聚集了子程序和中断堆栈。调用子程序和使能中断之前首先要定义堆栈空间，而且堆栈指针必须指向高于 0xFF 的地址空间。使用 PUSH 指令将数据推入堆栈时指针减一；而子程序或中断返回地址推入堆栈时指针将减二。使用 POP 指令将数据弹出堆栈时，堆栈指针加一；而用 RET 或 RETI 指令从子程序或中断返回时堆栈指针加二。

AVR的堆栈指针由I/O空间中的两个8位寄存器实现。实际使用的位数与具体器件有关。注意到某些 AVR 器件的数据区太小，用 SPL 就足够了。此时将不给出 SPH 寄存器。

位	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

指令执行时序

这一节介绍指令执行过程中的访问时序。AVR CPU 由系统时钟 clk_{CPU} 驱动。此时钟直接来自选定的时钟源。芯片内部不对此时钟进行分频。

Figure 6 说明了由 Harvard 结构决定的并行取指和指令执行，以及可以进行快速访问的寄存器文件的概念。这是一个性能高达 1 MIPS/MHz，具有优良的性价比、功能 / 时钟比、功能 / 功耗比的基本的流水线概念。

Figure 6. 并行取指和指令执行

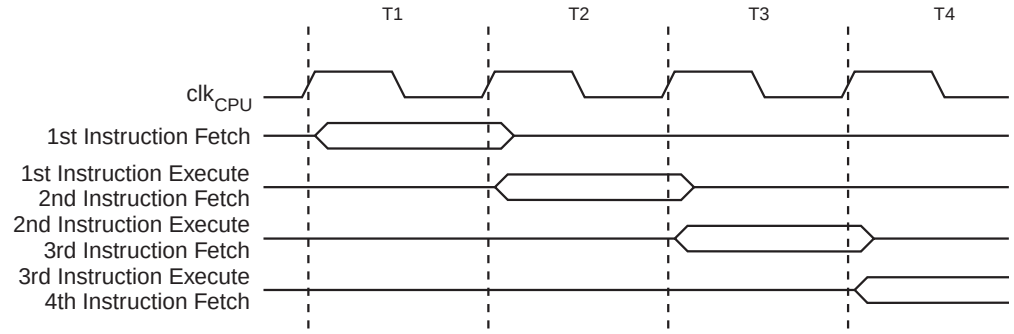
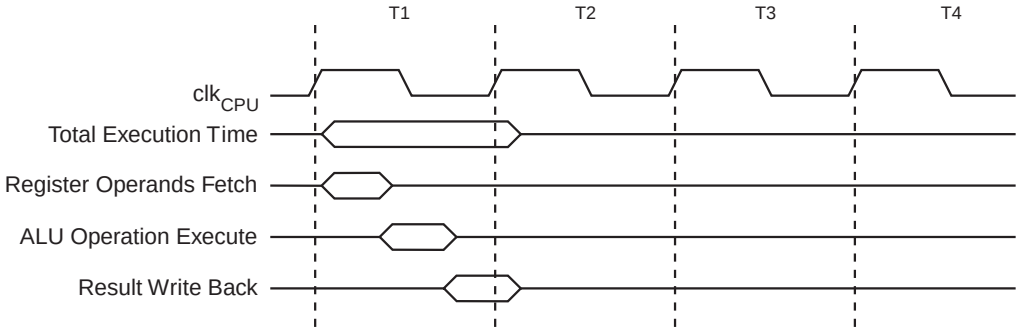


Figure7 演示的是寄存器文件内部访问时序。在一个时钟周期里，ALU 可以同时对两个寄存器操作数进行操作，同时将结果存回到目的寄存器中去。

Figure 7. 单时钟周期 ALU 操作



复位与中断处理

AVR 有不同的中断源。每个中断和复位在程序空间都有自己独立的中断向量。所有中断事件都有自己的使能位。在使能位置位，且状态寄存器的全局中断使能位 I 也置位的情况下，中断可以发生。根据不同的程序计数器 PC 数值，在引导 (Boot) 锁定位 BLB02 或 BLB12 被编程的情况下，中断可能自动禁止。这个特性提高了软件的安全性。详见 P255” 存储器编程 ” 部分。

程序存储区的最低地址缺省为复位向量和中断向量。完全的向量列表请参见 P48”中断 ”。列表也决定了不同中断的优先级。向量所在的地址越低，优先级越高。 RESET 具有最高的优先级，下一个则为 INT0 – 外部中断请求 0。通过置位 MCU 控制寄存器 MCUCR 的 IVSEL (具体参见 P48” 中断 ”), 中断向量可以移至引导 Flash 的起始处。编程熔丝位 BOOTRST 可以将复位向量也移至引导 Flash 的起始处。具体参见 P241”Boot Loader 支持 RWW 自编程， ATmega88 与 ATmega168” 。

任一中断发生时全局中断使能位 I 被清零，所有其他中断都被禁止。用户软件可以通过置位 I 来实现中断嵌套。此时所有的中断都可以中断当前中断服务程序。执行 RETI 指令后全局中断使能位 I 自动置位。

从根本上说有两种类型的中断。第一种由事件触发并置位中断标志。对于这些中断，程序计数器跳转到实际的中断向量以执行中断处理例程，同时硬件将清除相应的中断标志。中

断标志也可以通过对其写 ”1” 来清除。当中断发生后，如果相应的中断使能位为 "0"，则中断标志位置位，并一直保持到中断执行，或者被软件清除。类似的，如果全局中断标志被清零，则所有已发生的中断都不会被执行，直到 I 置位。然后被挂起的各个中断按中断优先级依次执行。

第二种类型的中断则是只要中断条件满足，就会一直触发。这些中断不需要中断标志。若中断条件在中断使能之前就消失了，中断不会被触发。

AVR 退出中断后总是回到主程序并至少执行一条指令才可以去执行其他被挂起的中断。

要注意的是，进入中断服务程序时状态寄存器不会自动保存；中断返回时也不会自动恢复。这些工作必须由用户通过软件来完成。

使用 CLI 指令来禁止中断时，中断禁止立即生效。没有中断可以在执行 CLI 指令后发生，即使它是在执行 CLI 指令的同时发生的。下面的例子说明了如何在写 EEPROM 时使用这个指令来防止中断发生以避免对 EEPROM 内容的可能破坏。

汇编代码例程

```
in  r16, SREG      ; 保存SREG
cli     ; 禁止中断
sbi EECR, EEMPE    ; 启动 EEPROM 写操作
sbi EECR, EEPE
out SREG, r16      ; 恢复SREG (I 位)
```

C 代码例程

```
char cSREG;
cSREG = SREG;  /* 保存SREG */
/* 禁止中断 */
_CLI();
EECR |= (1<<EEMPE); /* 启动 EEPROM 写操作 */
EECR |= (1<<EEPE);
SREG = cSREG; /* 恢复SREG (I 位) */
```

使用 SEI 指令使能中断时，紧跟其后的第一条指令在执行任何中断之前一定会首先得到执行。详见下面的例程：

汇编代码例程

```
sei   ; 置位全局中断使能标志
sleep ; 进入休眠模式，等待中断发生
; 注意： 在执行任何被挂起的中断之前 MCU 将首先进入休眠模式
;
```

C 代码例程

```
_SEI(); /* 置位全局中断使能标志 */
_SLEEP(); /* 进入休眠模式，等待中断发生 */
/* 注意： 在执行任何被挂起的中断之前 MCU 将首先进入休眠模式 */
```

中断响应时间

AVR 中断响应时间最少为 4 个时钟周期。4 个时钟周期后，程序跳转到实际的中断处理例程。在这 4 个时钟期间，PC 自动入栈。在通常情况下，中断向量为一个跳转指令，此跳转要花 3 个时钟周期。如果中断在一个多时钟周期指令执行期间发生，则在此多周期指令执行完毕后 MCU 才会执行中断程序。若中断发生时 MCU 处于休眠模式，中断响应时间还需增加 4 个时钟周期。此外还要考虑到不同的休眠模式所需要的启动时间。这个时间不包括在前面提到的时钟周期里。

中断返回需要 4 个时钟。在此期间 PC(两个字节) 将被弹出栈，堆栈指针加二，状态寄存器 SREG 的 I 置位。

ATmega48/88/168 的存储器

本节讲述 ATmega48/88/168 的存储器。 AVR 结构具有两个主要的存储器空间：数据存储器空间和程序存储器空间。此外， ATmega48/88/168 还有 EEPROM 存储器以保存数据。这三个存储器空间都为线性的平面结构。

系统内可编程的 Flash 程序存储器

ATmega48/88/168 具有 4/8/16K 字节的在线编程 Flash，用于存放程序指令代码。因为所有的 AVR 指令为 16 位或 32 位，故而 Flash 组织成 2/4/8K x 16. 对于 ATmega88 与 ATmega168，用户程序的安全性要根据 Flash 程序存储器的两个区：引导 (Boot) 程序区和应用程序区分开来考虑。ATmega48 中没有分为 引导程序区和应用程序区，SPM 指令可在整个 Flash 中执行。详见 P236” 存贮程序存储器 (SPM) 控制和状态寄存器 – SPMCSR” 与 P244 中 SPMEN 部分的描述。

Flash 存储器至少可以擦写 10,000 次。ATmega48/88/168 的程序计数器 (PC) 为 11/12/13 位，因此可以寻址 2/4/8K 的程序存储器空间。引导程序区以及相关的软件安全锁定位见 P234”Flash自编程，ATmega48” 与P241”Boot Loader支持RWW自编程，ATmega88 与 ATmega168” 。而 P255”存储器编程” 详述了用SPI引脚或JTAG接口实现对Flash数据的串行下载。

常数可以保存于整个程序存储器地址空间 (参考 LPM 加载程序存储器指令说明)。

取指与执行时序图请参见 P11” 指令执行时序 ” 。

Figure 8. ATmega48 的程序存储器映像

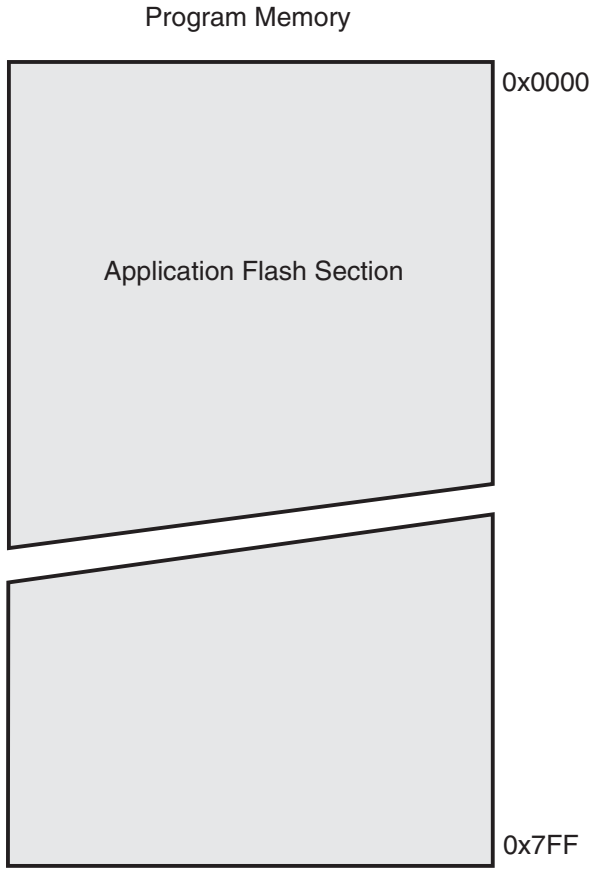
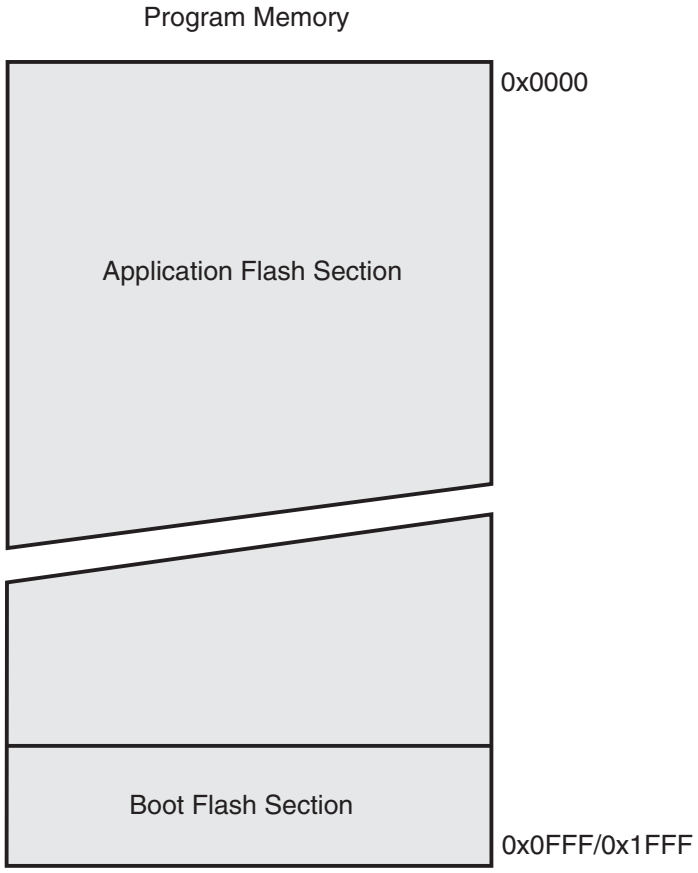


Figure 9. ATmega88 与 ATmega168 的程序存储器映像



SRAM 数据存储器

Figure10 给出了 ATmega48/88/168 SRAM 空间的组织结构。

ATmega48/88/168 是一个复杂的微控制器，其支持的外设要比预留的 64 个 I/O(通过 IN/OUT 指令访问) 所能支持的要多。对于扩展的 I/O 空间段 0x60 - 0xFF 只能使用 ST/STS/STD 和 LD/LDS/LDD 指令。

前 768/1280/1280 个数据存储器包括了寄存器文件， I/O 存储器，扩展的 I/O 存储器以及数据 SRAM。起始的 32 个地址为寄存器文件，然后是 64 个 I/O 存储器，接着是 160 个扩展 I/O 存储器。最后是 512/1024/1024 字节的数据 SRAM。

数据存储器的寻址方式分为 5 种 : 直接寻址、带偏移量的间接寻址、间接寻址、预减间接寻址和后加间接寻址。寄存器文件中的寄存器 R26 到 R31 为间接寻址的指针寄存器。

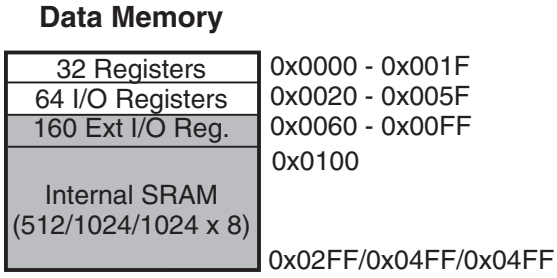
直接寻址范围可达整个数据区。

带偏移量的间接寻址模式能够寻址到由寄存器 Y 和 Z 给定的基址附近的 63 个地址。

在自动预减和后加的间接寻址模式中，寄存器 X、 Y 和 Z 自动增加或减少。

ATmega48/88/168 的全部 32 个通用寄存器、64 个 I/O 寄存器、160 个扩展 I/O 寄存器及 512/1024/1024 个字节的内部数据 SRAM 可以通过所有上述的寻址模式进行访问。寄存器文件的描述见 P8” 通用寄存器文件 ” 。

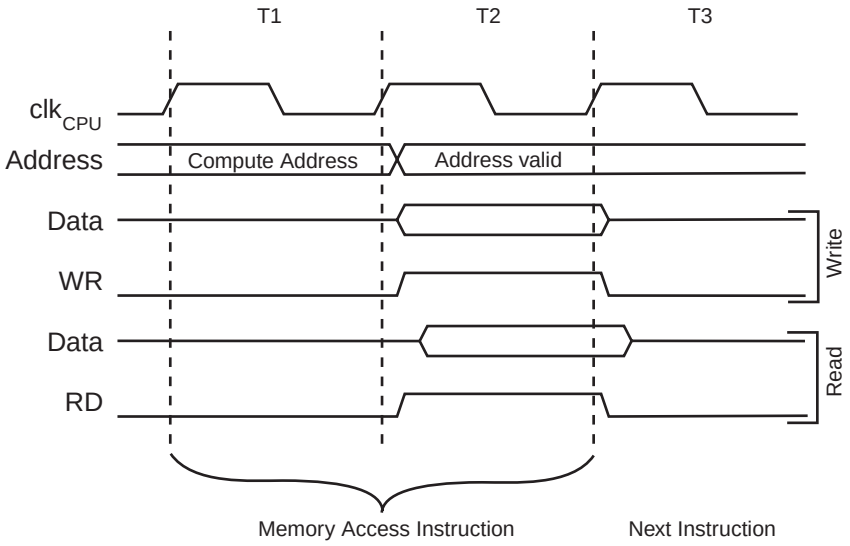
Figure 10. 数据存储器映像



数据存储器访问时间

本节说明访问内部存储器的时序。如 Figure11 所示，内部数据 SRAM 访问时间为两个 clk_{CPU} 时钟。

Figure 11. 片上 SRAM 存取周期



EEPROM 数据存储器

ATmega48/88/168 包含 256/512/512 字节的 EEPROM 数据存储器。它是作为一个独立的数据空间而存在的，可以按字节读写。 EEPROM 的寿命至少为 100,000 次擦除周期。EEPROM 的访问由地址寄存器，数据寄存器和控制寄存器决定。

具体的 SPI 及并行下载 EEPROM 数据请参见 P255” 存储器编程 ”

EEPROM 读 / 写访问

EEPROM 的访问寄存器位于 I/O 空间。

EEPROM 的写访问时间由 Table3 给出。自定时功能可以让用户监测何时开始写下一字节。如果用户要操作 EEPROM，应当注意如下问题：在电源滤波时间常数比较大的电路中，上电 / 下电时 V_{CC} 上升 / 下降速度会比较慢。此时 CPU 将工作于低于晶振所要求的电源电压。请参见 P21”防止EEPROM数据丢失” 以避免出现EEPROM的数据丢失问题。

为了防止无意识的 EEPROM 写操作，在写 EEPROM 时需要执行一个特定的写时序。具体参看 EEPROM 控制寄存器的内容。

当执行 EEPROM 读操作时， CPU 会停止工作 4 个周期，然后再执行后续指令；当执行 EEPROM 写操作时， CPU 会停止工作 2 个周期，然后再执行后续指令。

EEPROM 地址寄存器—EEARH 和 EEARL

位	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

- **位 15..9 – Res: 保留**

保留位，读操作返回值为零 。

- **位 8..0 – EEAR8..0: EEPROM 地址**

EEPROM 地址寄存器 – EEARH 和 EEARL 指定了 256/512/512 字节的 EEPROM 空间。EEPROM 地址是线性的，从 0 到 255/511/511。EEAR 的初始值没有定义。在访问 EEPROM 之前必须为其赋予正确的数据。

EEAR8 在 ATmega48 中为无效位，必须始终将其赋值为 ”0”。

EEPROM 数据寄存器—EEDR

位	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..0 – EEDR7..0: EEPROM 数据**

对于 EEPROM 写操作，EEDR 是需要写到 EEAR 单元的数据；对于读操作，EEDR 是从地址 EEAR 读取的数据。

EEPROM 控制寄存器—EECR

位	7	6	5	4	3	2	1	0	
	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	X	X	0	0	X	0	

- **位 7..4 – Res: 保留**

保留位，读操作返回值为零。

- **位 5, 4 – EEPM1 与 EEPM0: EEPROM 编程模式位**

EEPROM 编程模式位的设置决定对 EEPE 写入后将触发什么编程方式。EEPROM 的编程可以作为一个原子操作来实现（擦除老的数据并写入新的数据），也可以将擦除与写操作分为两步进行。不同编程模式的时序请见 Table2。EEPE 置位时，对 EEPMn 的任何写操作都将会被忽略。在复位过程中，除非 EEPROM 处于编程状态，EEPMn 位将被设置为 0b00。

Table 2. EEPROM 模式位

EEPM1	EEPM0	编程时间	操作
0	0	3.4 ms	擦与写在一个操作中完成 (基本操作)
0	1	1.8 ms	只擦操作
1	0	1.8 ms	只写操作
1	1	–	保留

- **位 3 – EERIE: 使能 EEPROM 准备好中断**

若 SREG 的 I 为 "1"，则置位 EERIE 使能 EEPROM 准备好中断。清零 EERIE 则禁止此中断。当 EEWE 清零时 EEPROM 准备好中断即可发生。

- **位 2 – EEMWE: EEPROM 写使能**

EEMWE决定设置EEWE为"1"是否可以启动EEPROM写操作。。当EEMWE为"1"时，在4个时钟周期内置位 EEWE 将把数据写入 EEPROM 的指定地址；若 EEMWE 为 "0"，则 EEWE 不起作用。EEMWE 置位后 4 个周期，硬件对其清零。见 EEPROM 写过程对 EEPE 位的描述。

- **位 1 – EEPE: EEPROM 写使能**

写使能信号 EEPE 是 EEPROM 的写入选通信号。当 EEPROM 数据和地址设置好之后，需置位 EEPE 以便将数据写入 EEPROM。此时 EEMPE 必须置位，否则 EEPROM 写操作将不会发生。写时序如下 (第 3 和第 4 步的次序可更改) :

1. 等待 EEPE 为 ”0”
2. 等待 SPMCSR 寄存器的 SPMEN 为零
3. 将新的 EEPROM 地址写入 EEAR (可选)
4. 将新的 EEPROM 数据写入 EEDR (可选)
5. 对 EECR 寄存器的 EEMPE 写 "1"，同时清零 EEPE
6. 在置位 EEMPE 之后的 4 个周期内置位 EEPE

在 CPU 写 Flash 存储器的时候不能对 EEPROM 进行编程。在启动 EEPROM 写操作之前软件必须要检查 Flash 写操作是否已经完成。第二步仅在软件包含引导程序，允许 CPU 对 Flash 进行编程时才有用。如果 CPU 永远都不会写 Flash，则第二步可以忽略。请参考 P241”Boot Loader 支持 RWW 自编程， ATmega88 与 ATmega168” 。

注意 : 如有中断发生于步骤 5 和 6 之间将导致写操作失败。因为此时 EEPROM 写使能操作将超时。如果一个操作 EEPROM 的中断打断了另一个 EEPROM 操作，EEAR 或 EEDR 寄存器可能被修改，引起 EEPROM 操作失败。建议此时关闭全局中断标志 I。

经过写访问时间之后，EEPE 硬件清零。用户可以凭此位判断写时序是否已经完成。EEPE 置位后， CPU 要停止两个时钟周期才会运行下一条指令。

- **位 0 – EERE: EEPROM 读使能**

读使能信号 EERE 是 EEPROM 的写入选通信号。当 EEPROM 地址设置好之后，需置位 EERE 以便将数据读入 EEAR。EEPROM 数据的读取只需要一条指令。读取 EEPROM 时 CPU 要停止 4 个时钟周期然后才能执行下一条指令。

用户在读取 EEPROM 时应该检测 EEPE。如果一个写操作正在进行，就无法读取 EEPROM，也无法改变寄存器 EEAR。

标定的振荡器用于 EEPROM 访问定时。Table3 为 CPU 访问 EEPROM 的典型时间。

Table 3. EEPROM 编程时间

符号	标定的 RC 振荡器周期数	典型编程时间
CPU 发起的 EEPROM 写操作	26,368	3.3 ms

下面的代码分别用汇编和 C 函数说明如何实现 EEPROM 的写操作。在此假设中断不会在执行这些函数的过程当中发生。例子同时还假设软件没有引导程序。若引导程序存在，则 EEPROM 写函数还需要等待正在进行的 SPM 命令的结束。

汇编代码例程

```
EEPROM_write:
  ; 等待上一次写操作结束
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; 设置地址寄存器 (r18:r17)
  out  EEARH, r18
  out  EEARL, r17
  ; 写数据到数据寄存器 (r16)
  out  EEDR,r16
  ; 置位 EEMWE
  sbi  EECR,EEMWE
  ; 置位 EEWE 以启动写操作
  sbi  EECR,EEWE
  ret
```

C 代码例程

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEWE))
    ;
  /* 设置地址和数据寄存器 */
  EEAR = uiAddress;
  EEDR = ucData;
  /* 置位 EEMWE */
  EECR |= (1<<EEMWE);
  /* 置位 EEWE 以启动写操作E */
  EECR |= (1<<EEWE);
}
```

下一个代码例子说明如何用汇编和 C 来读取 EEPROM。在此假设中断不会在执行这些函数的过程当中发生。

汇编代码例程

```
EEPROM_read:
  ; 等待上一次写操作结束
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; 设置地址寄存器 (r18:r17)
  out  EEARH, r18
  out  EEARL, r17
  ; 设置EERE 以启动读操作
  sbi  EECR,EERE
  ; 自数据寄存器读取数据
  in   r16,EEDR
  ret
```

C 代码例程

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEWE))
    ;
  /* 设置地址寄存器 */
  EEAR = uiAddress;
  /* 设置EERE 以启动读操作 */
  EECR |= (1<<EERE);
  /* 自数据寄存器返回数据 */
  return EEDR;
}
```

防止 EEPROM 数据丢失

如果电源电压过低，CPU 和 EEPROM 有可能工作不正常，造成 EEPROM 数据的毁坏 (丢失)。这种情况在使用独立的 EEPROM 器件时也会遇到。

由于电压过低造成 EEPROM 数据损坏有两种可能：一是电压低于 EEPROM 写操作所需要的最低电压；二是 CPU 本身已经无法正常工作。

EEPROM 数据损坏的问题可以通过以下方法来避免：

当电压过低时保持 AVR RESET 信号为低。这可以通过使能芯片的掉电检测电路 BOD 来实现。如果 BOD 电平无法满足要求，则可以使用外部复位电路。若写操作过程当中发生了复位，若电源电压足够高，写操作仍将正常结束。

I/O 存储器

ATmega48/88/168 的 I/O 空间定义见 P311” 寄存器概述 ” 。

ATmega48/88/168 的所有 I/O 和外设都被放置在 I/O 空间。所有的 I/O 地址都可以通过 LD/LDS/LDD和ST/STS/STD指令来访问，在32个通用工作寄存器和I/O之间传输数据。地址为 0x00 - 0x1F 的 I/O 寄存器还可用 SBI 和 CBI 指令直接进行位寻址，而 SBIS 和 SBIC 则用来检查单个位置位与否。使用 IN 和 OUT 指令时地址必须在 0x00 - 0x3F 之间。如果要象 SRAM 一样通过 LD 和 ST 指令访问 I/O 寄存器，相应的地址要加上 0x20。ATmega48/88/168 是一个复杂的微处理器，其支持的外设要比预留的 64 个 I/O(通过 IN/OUT 指令访问) 所能支持的要多。对于扩展的 I/O 空间 0x60 - 0xFF，只能使用 ST/STS/STD 和 LD/LDS/LDD 指令。

为了与后续产品兼容，保留未用的位应写 "0“，而保留的 I/O 寄存器则不应进行写操作。

一些状态标志位的清除是通过写 "1“ 来实现的。CBI 和 SBI 指令可以操作 I/O 寄存器所有的位，并给置位的位回写 "1“，因此会清除这些标志位。CBI 和 SBI 指令只对 0x00 to 0x1F 之间的寄存器有效。

I/O 和外设控制寄存器在其他章节介绍。

通用 I/O 寄存器

ATmega48/88/168包含3个通用I/O寄存器。这些寄存器可以用来存储信息，尤其适合于存储全局变量与状态标志。位于 0x00 - 0x1F 的通用 I/O 寄存器可以通过 SBI、 CBI、 SBIS 与 SBIC 指令直接进行位寻址。

通用 I/O 寄存器 2—GPIOR2

位	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通用 I/O 寄存器 1—GPIOR1

位	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR1
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通用 I/O 寄存器 0—GPIOR0

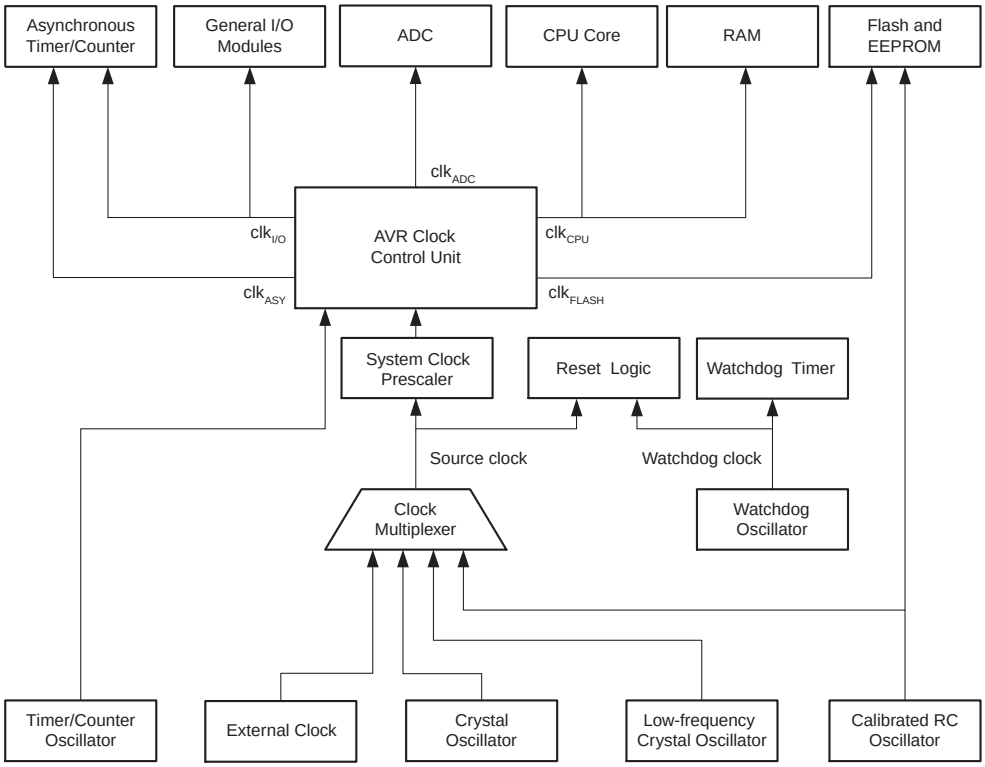
位	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

系统时钟及其选项

时钟系统及其分布

Figure12 为 AVR 的主要时钟系统及其分布。这些时钟并不需要同时工作。为了降低功耗，可以通过使用不同的睡眠模式来禁止无需工作的模块的时钟，详见 P32” 电源管理及休眠模式 ”。下面为时钟系统的详细描述。

Figure 12. 时钟分布



CPU 时钟—clk_{CPU}

CPU时钟与操作AVR内核的子系统相连，如通用寄存器文件、状态寄存器及保存堆栈指针的数据存储器。终止 CPU 时钟将使内核停止工作和计算。

I/O 时钟—$clk_{I/O}$

I/O 时钟用于主要的 I/O 模块，如定时器 / 计数器、SPI 和 USART。I/O 时钟还用于外部中断模块。要注意的是有些外部中断通过异步逻辑检测，因此即使 I/O 时钟停止了这些中断仍然可以得到监控。此外， USI 模块的起始条件检测在没有 $clk_{I/O}$ 的情况下也是异步实现的，使得这个功能在任何睡眠模式下都可以正常工作。

Flash 时钟—clk_{FLASH}

Flash 时钟控制 Flash 接口的操作。此时钟通常与 CPU 时钟同时挂起或激活。

异步定时器时钟—clk_{ASY}

异步定时器时钟允许异步定时器 / 计数器直接由外部 32 kHz 时钟晶体驱动。使得此定时器 / 计数器即使在睡眠模式下仍然可以为系统提供一个实时时钟。

ADC 时钟—clk_{ADC}

ADC具有专门的时钟。这样可以在ADC工作的时候停止CPU和I/O时钟以降低数字电路产生的噪声，从而提高 ADC 转换精度。

时钟源

芯片有如下几种通过 Flash 熔丝位选择的时钟源。时钟输入到 AVR 时钟发生器，再分配到相应的模块。

Table 4. 时钟源选择 (1)

器件时钟选项	熔丝位 CKSEL3..0
低功耗晶振	1111 - 1000
满振幅晶振	0111 - 0110
低频晶振	0101 - 0100
内部 128 kHz RC 振荡器	0011
校准的内部 RC 振荡器	0010
外部时钟	0000
保留	0001

Note: 1. 对于所有的熔丝位，“1” 表示未编程，“0” 代表已编程。

缺省时钟源

器件出厂时内部 RC 振荡器频率标定为 8.0MHz 并且 CKDIV8 被编程，得到 1.0MHz 的系统时钟。启动时间设为最长且定时周期使能。(CKSEL = "0010"，SUT = "10"， CKDIV8 = "0")。这种设置保证用户可以通过任何有效的编程接口获得需要的时钟源。

时钟启动序列

任何时钟源都需要足够高的 Vcc 来启动振荡器，并且还要有一个最小振荡周期以保证在开始正常工作之前电源达到稳定电平 。

为保证有足够高的 Vcc，在其它复位源释放之后，器件在一个超时延迟时间 (t_{TOUT}) 之内保持内部复位状态。P37”系统控制和复位” 描述了内部复位的开始条件。这个延时(t_{TOUT})由看门狗振荡器定时，而延时的周期数通过熔丝位 SUTx 与 CKSELx 来设定。Table5 列出了可选的延迟。看门狗振荡器的频率由工作电压决定，详见 P283”ATmega48/88/168 典型特性 – 初始数据 ” 。

Table 5. 看门狗振荡器周期数

典型的溢出时间 (V_{CC} = 5.0V)	典型的溢出时间 (V_{CC} = 3.0V)	时钟周期数
0 ms	0 ms	0
4.1 ms	4.3 ms	4K (4,096)
65 ms	69 ms	8K (8,192)

延时的主要目的是保证在系统能够提供符合应用要求的最小的 Vcc 之前 AVR 处于复位状态。延时过程之中 MCU 并不监控实际的电压。因此需要用户选择合适的、长于 Vcc 上升时间的延时时间。如果无法做到这一点，就应该使用内 / 外部 BOD。BOD 电路可以保证在释放复位之前有足够高的 Vcc。使用 BOD 时可以禁止超时延迟。我们不推荐在没有使用 BOD 电路时禁止超时延迟的设计。

在时钟进入稳定状态前，振荡器需要振荡几个周期。芯片内部有一个纹波计数器监控振荡器输出时钟，并且在达到给定周期数之前保证内部复位有效。然后计数器释放复位信号，器件开始执行程序。推荐的振荡启动时间与时钟类型有关，可以是外部时钟的 6 个周期到低频晶振的 32K 个周期。

当芯片从复位状态启动时，时钟的起始序列包括超时延时与启动时间。当 CPU 自掉电模式或省电模式唤醒之后， Vcc 被认为足够高，因此起始序列只包括了启动时间。

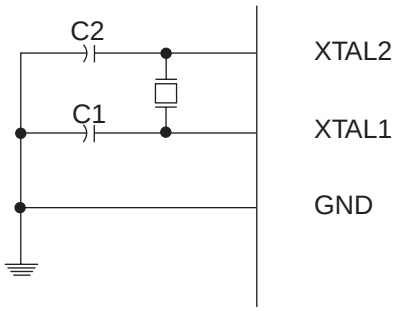
低功率晶振

XTAL1 与 XTAL2 引脚分别是片内振荡器的反向放大器输入、输出端 ，见 Figure13。这个振荡器可以使用石英晶体，也可以使用陶瓷谐振器。

该振荡器是一个低功率振荡器， XTAL2 输出电压的摆幅比平常的要低。它提供了最低的功耗，但不能驱动其它的时钟输入，在噪声环境中也更易受影响，具体请参见 P26” 满振幅晶振 ”。

电容 C1、 C2 的值总是相等的。具体电容值的选择取决于使用的是石英晶体还是陶瓷振荡器，以及总的杂散电容与环境电磁噪声等。Table6 给出了采用石英晶体时的电容选择范围。使用陶瓷振荡器时，电容值应采用生产商给出的值。

Figure 13. 晶体振荡器连接图



振荡器可以工作于三种不同的模式，每一种都有一个优化的频率范围。工作模式通过熔丝位 CKSEL3..1 来选择，如 Table6 所示。

Table 6. 低功率晶体振荡器工作模式 (3)

频率范围 (1) (MHz)	CKSEL3..1	使用晶体时电容 C1 和 C2 的推荐范围 (pF)
0.4 - 0.9	100(2)	–
0.9 - 3.0	101	12 - 22
3.0 - 8.0	110	12 - 22
8.0 - 16.0	111	12 - 22

Notes: 1. 频率范围只是初步值，实际值待测。
2. 此选项不适用于晶体，只能用于陶瓷谐振器。
3. 如果8 MHz频率超出器件的规格 (由V_{CC}决定)，可编程熔丝位CKDIV8将内部频率 8分频。但必须保证所得时钟符合芯片的频率要求。

如 Table7 所示，熔丝位 CKSEL0 以及 SUT1..0 用于选择启动时间。

Table 7. 低功率晶体振荡器时钟选项对应的启动时间

振荡源 / 电源状态	掉电与节电模式下的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	CKSEL0	SUT1..0
陶瓷谐振器，电源快速上升	258 CK	14CK + 4.1 ms(1)	0	00
陶瓷谐振器，电源缓慢上升	258 CK	14CK + 65 ms(1)	0	01
陶瓷谐振器，BOD 使能	1K CK	14CK(2)	0	10
陶瓷谐振器，电源快速上升	1K CK	14CK + 4.1 ms(2)	0	11
陶瓷谐振器，电源缓慢上升	1K CK	14CK + 65 ms(2)	1	00
石英振荡器，BOD 使能	16K CK	14CK	1	01
石英振荡器，电源快速上升	16K CK	14CK + 4.1 ms	1	10
石英振荡器，电源慢速上升	16K CK	14CK + 65 ms	1	11

Notes: 1. 这些选项只能用于工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要的情况。不适用于晶体。
2. 这些选项是为陶瓷谐振器设计的，可以保证启动时频率足够稳定。若工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要时也可以适用于晶体。

满振幅晶振

XTAL1 与 XTAL2 引脚分别是片内振荡器的反向放大器输入、输出端 ，见 Figure13。这个振荡器可以使用石英晶体，也可以使用陶瓷谐振器。

该振荡器为满振幅振荡器， XTAL2 引脚的输出为满幅振荡信号。可用来驱动其它的时钟输入端，且可在噪声环境中工作。电流消耗大于 P25” 低功率晶振 ”。注意满振幅晶振只在 Vcc = 2.7 - 5.5V 时可用。

电容 C1、 C2 的值总是相等的。具体电容值的选择取决于使用的是石英晶体还是陶瓷振荡器，及总的杂散电容与环境电磁噪声等。 Table9 给出了采用石英晶体时的电容选择范围。使用陶瓷振荡器时，电容值应采用生产商给出的值。

工作模式通过熔丝位 CKSEL3..1 来选择，如 Table8 所示。

Table 8. 满振幅晶体振荡器工作模式 (2)

频率范围 (1) (MHz)	CKSEL3..1	使用晶体时电容 C1 和 C2 的推荐范围 (pF)
0.4 - 25	111	12 - 22

Notes: 1. 频率范围只是初步值，实际值待测。
2. 如果8 MHz频率超出器件的规格 (由V_{CC}决定)，可对熔丝位CKDIV8编程将内部频率 8 分频。但必须保证所得时钟符合器件的频率要求。

Figure 14. 晶体振荡器连接图

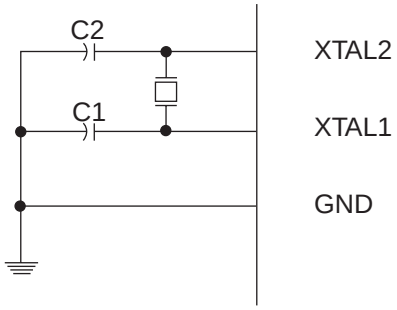


Table 9. 满振幅晶体振荡器时钟选项对应的启动时间

振荡源 / 电源状态	掉电与节电模式下的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	CKSEL0	SUT1..0
陶瓷谐振器，电源快速上升	258 CK	14CK + 4.1 ms(1)	0	00
陶瓷谐振器，电源缓慢上升	258 CK	14CK + 65 ms(1)	0	01
陶瓷谐振器，BOD 使能	1K CK	14CK(2)	0	10
陶瓷谐振器，电源快速上升	1K CK	14CK + 4.1 ms(2)	0	11
陶瓷谐振器，电源缓慢上升	1K CK	14CK + 65 ms(2)	1	00
石英谐振器，BOD 使能	16K CK	14CK	1	01
石英振荡器，电源快速上升	16K CK	14CK + 4.1 ms	1	10
石英振荡器，电源慢速上升	16K CK	14CK + 65 ms	1	11

Notes: 1. 这些选项只能用于工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要的情况。不适用于晶体。
2. 这些选项是为陶瓷谐振器设计的，可以保证启动时频率足够稳定。若工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要时也可以适用于晶体。

低频晶振

可以使用外部 32.768 kHz 表用振荡器作为器件的时钟源。晶体连接方式见 Figure13。选定振荡器后，启动时间由 Table10 中列出的 SUT 熔丝位与 CKSEL0 决定。

Table 10. 低频晶体振荡器的启动时间

电源状态	掉电与节电模式下的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	CKSEL0	SUT1..0
BOD 使能	1K CK	14CK[1]	0	00
电源快速上升	1K CK	14CK + 4.1 ms[1]	0	01
电源缓慢上升	1K CK	14CK + 65 ms[1]	0	10
保留			0	11
BOD 使能	32K CK	14CK	1	00
电源快速上升	32K CK	14CK + 4.1 ms	1	01
电源缓慢上升	32K CK	14CK + 65 ms	1	10
保留			1	11

Note: 1. 这些选项只能用于启动时的频率稳定性对于应用而言不重要的情况。

标定的片内 RC 振荡器

校准的片内 RC 振荡器提供了固定的 8.0 MHz 的时钟，这是在 3V、25°C 下的标称数值。器件出厂时 CKDIV8 熔丝位已经被编程，请参见 P31” 系统时钟预分频器 ” 。按照 Table11 对熔丝位 CKSEL 进行编程即可将其作为系统时钟。选择这个时钟之后就无需外部器件了。复位时硬件将标定字节加载到 OSCCAL 寄存器，自动完成对 RC 振荡器的标定。在 3V、25°C 时，这种标定可以提供标称频率 ± 1% 的精度。通过改变 OSCCAL 寄存器，标定可以使振荡器在 7.3 - 8.1 MHz 的范围内的精度达到 ± 1%。当使用这个振荡器作为系统时钟时，看门狗振荡器继续为看门狗定时器和溢出复位提供时钟。更多的有关标定数据的信息请参见 P259” 校准字节 ” 。

Table 11. 片内标定的 RC 振荡器工作模式 [1][3]

频率范围 [2] (MHz)	CKSEL3..0
7.3 - 8.1	0010

Notes: 1. 出厂时的设置
2. 频率范围只是初步值，实际值待测
3. 如果8 MHz频率超出器件的规格 (由V_{CC}决定)，可对熔丝位CKDIV8编程将内部频率 8 分频

选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 P28Table12 所示。

Table 12. 内部标定 RC 振荡器的启动时间

电源状态	掉电与节电模式下的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	SUT1..0
BOD 使能	6 CK	14CK[1]	00
电源快速上升	6 CK	14CK + 4.1 ms	01
电源缓慢上升	6 CK	14CK + 65 ms[2]	10
保留			11

Note: 1. 如果RSTDISBL熔丝位被编程 ,启动时间将增加到14CK + 4.1 ms以保证能够进入编程模式

2. 出厂时的设置

振荡器标定寄存器—OSCCAL

位	7	6	5	4	3	2	1	0	
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	标定数据								

- **位 7..0 – CAL7..0: 振荡器标定数据**

将标定数据写入这个地址可以对内部振荡器进行调节以消除由于生产工艺所带来的振荡器频率偏差。这在芯片复位时自动完成。 25°C 时振荡器频率为 8.0 MHz。应用软件可对该寄存器进行写操作来改变振荡器频率。振荡器频率范围在 7.3 - 8.1 MHz 内标定精度可达到 ±1% 以内，频率超出此范围则无法保证。

注意该振荡器用来为 EEPROM 与 Flash 的写访问定时，从而对相应的时序产生影响。如果需要对 EEPROM 或 Flash 执行写入操作，不要将频率标定到高于 8.8 MHz。否则对 EEPROM 或 Flash 的写操作将会失败。

CAL7位决定振荡器工作范围。将该位置0给出低端频率范围 ,而将该位置1则给出高端频率范围。这两个频率范围是有重叠的，也就是说 OSCCAL = 0x7F 给出的频率高于 OSCCAL = 0x80。

CAL6..0位 用来调节被选中范围内的频率。设置为0x00表示该范围中的最低频率 ,设置为0x7F表示该范围中的最高频率。当频率范围在7.3 - 8.1 MHz时 ,CAL6..0中的值增加1频率值增加低于 2%。

128 kHz 内部振荡器

128 kHz 内部振荡器是低功率振荡器 ,在3V、25°C条件下的标称频率为128 kHz。CKSEL熔丝位编程为 “11” 时该时钟被选为系统时钟，如 Table13 所示。

Table 13. 128 kHz 内部振荡器工作模式

标称频率	CKSEL3..0
128 kHz	0011

Note: 1. 频率范围只是初步值，实际值待测

选择了时钟源之后，启动时间由熔丝位 SUT 确定，如 Table14 所示。

Table 14. 128 kHz 内部振荡器的启动时间

电源状态	掉电与节电模式下的启动时间	复位时的额外延迟时间	SUT1..0
BOD 使能	6 CK	14CK(1)	00
电源快速上升	6 CK	14CK + 4 ms	01
电源缓慢上升	6 CK	14CK + 64 ms	10
保留			11

Note: 1. 如果RSTDISBL熔丝位被编程，启动时间将增加到14CK + 4.1 ms以保证能够进入编程模式

外部时钟

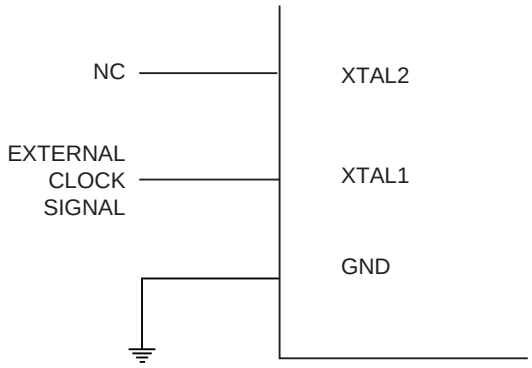
芯片可用外部时钟源驱动，如Figure15所示。此时CKSEL熔丝位必须按照Table15编程。

Table 15. 满振幅晶振工作模式 (2)

频率范围 (1) (MHz)	CKSEL3..0	使用晶体时电容 C1 和 C2 的推荐范围 (pF)
0 - 100	0000	12 - 22

Notes: 1. 频率范围只是初步值，实际值待测。
2. 如果8 MHz频率超出器件的规格 (由V_{CC}决定)，可对熔丝位CKDIV8编程将内部频率 8 分频。但必须保证所得时钟符合器件的频率要求。

Figure 15. 外部时钟配置



选择了这个振荡源之后，启动时间由熔丝位 SUT 确定，如 Table16 所示。

Table 16. 外部时钟的启动时间

电源状态	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	SUT1..0
BOD 使能	6 CK	14CK	00
电源快速上升	6 CK	14CK + 4.1 ms	01
电源缓慢上升	6 CK	14CK + 65 ms	10
保留			11

为了保证 MCU 能够稳定工作，不能突然改变外部时钟源的振荡频率。工作频率突变超过 2% 将产生异常现象。最好是在 MCU 保持复位状态时改变外部时钟的振荡频率。

要注意的是，系统时钟预分频可以实现在运行期间改变内部时钟频率而保持系统稳定运行。请参见 P31” 系统时钟预分频器 ” 。

时钟输出缓冲器

CKOUT 熔丝位编程后，系统时钟将从 CLKO 输出。这种模式适用于芯片时钟用来驱动系统内其它电路。即使芯片处于复位状态，此时钟还会被输出。对 CKOUT 熔丝位编程后 I/O口的正常操作被切换为时钟输出。当CLKO作为时钟输出时，系统时钟可以为包括片内 RC 振荡器在内的所有时钟源。如果使用系统时钟预分频，输出的是被分频后的系统时钟频率。

定时器 / 计时器振荡器

该器件可通过外部 32.768 kHz 表用振荡器或外部时钟源对定时器 / 计数器 2 进行操作 。定时器 / 计数器引脚 TOSC1/TOSC2 和 XTAL1 /XTAL2 是共用的。这意味着只有当内部标定 RC 振荡器作为系统时钟源时定时器 / 计数器振荡器才能使用。参见 P24Figure 13 的晶体振荡器连接图。

如果 ASSR 寄存器中的 EXTCLK 写入逻辑 ”1”，可以对 TOSC1 使用外部时钟源。如何想使用外部时钟而不是 32 kHz 的晶体，请参见 P139” 定时器 / 计数器的异步操作 ” 。

系统时钟预分频器

ATmega48/88/168 可以通过设置时钟预分频寄存器CLKPR来得到分频的系统时钟。当需要的系统处理能力比较低时可以利用这个特性来降低功耗。预分频对所有时钟源都适用，并且影响 CPU 及所有同步外设的时钟频率。 $clk_{I/O}$、clk_{ADC}、clk_{CPU} 和 clk_{FLASH} 则根据 Table17 所列出的系数进一步被分频。

时钟预分频寄存器—CLKPR

位	7	6	5	4	3	2	1	0	
	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPR
读 / 写	R/W	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	参见位说明				

- **位 7 – CLKPCE: 时钟预分频器更新使能**

如果要改变 CLKPS 的内容， CLKPCE 必须写入逻辑 "1”。只有当 CLKPR 中的其它位同时写入 "0” 时， CLKPCE 位才会更新。在 CLKPCE 被写入逻辑 "1” 之后 4 个时钟周期，或是 CLKPS 位被执行写操作之后 4 个时钟周期， CLKPCE 将被硬件清除。在这 4 个时钟周期的计时时间内改变 CLKPCE 既不会延长此计时时间，也不会清零 CLKPCE。

- **位 3..0 – CLKPS3..0: 时钟预分频选择位 3 - 0**

这几位确定了被选时钟源与内部系统时钟间的分频因子。它们在系统运行期间可以进行修改来改变时钟频率以适应实际需要。由于分频器分频的对象是 输入到 MCU 的主时钟，使用分频因子将改变所有同步外设的运行速度。分频因子在 Table17 中给出。

为避免无意中改变时钟频率，在改变 CLKPS 时必须执行一个特殊的写程序：

1. 将时钟预分频器更新使能 (CLKPCE) 置位，并同时将 CLKPR 的其余位清零
2. 在随后的 4 个时钟周期内，将需要的数值写入 CLKPS 并同时将 CLKPCE 清零

为保证写操作不被中断，在改变预分频设置时必须禁止中断。

熔丝位 CKDIV8 决定了 CLKPS 的初始值。如果 CKDIV8 未编程， CLKPS 将被设为 “0000”。如果CKDIV8已编程，CLKPS将被设为“0011”，在芯片启动时将8作为分频因子。如果选择的时钟源超出了器件工作条件所能允许的最大频率值，则一定要编程这个熔丝位。不论 CKDIV8 熔丝如何设定， CLKPS 的值都可以任意进行修改。因此当选择的时钟

源超出了器件工作条件所能允许的最大频率值时，应用软件必须使用合适的分频因子。CKDIV8 熔丝出厂时已编程。

Table 17. 时钟预分频选择

CLKPS3	CLKPS2	CLKPS1	CLKPS0	时钟分频因子
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256
1	0	0	1	保留
1	0	1	0	保留
1	0	1	1	保留
1	1	0	0	保留
1	1	0	1	保留
1	1	1	0	保留
1	1	1	1	保留

电源管理及休眠模式

休眠模式可以使应用程序关闭 MCU 中没有使用的模块，从而降低功耗。 AVR 具有不同的休眠模式，允许用户根据自己的应用要求实施剪裁。

进入 5 个休眠模式的条件是置位寄存器 SMCR 的 SE，然后执行 SLEEP 指令。具体哪一种模式 (空闲模式、ADC 噪声抑制模式、掉电模式、省电模式和 Standby 模式) 由 SMCR 的 SM2、SM1 和 SM0 决定，如 Table18 所示。使能的中断可以将进入休眠模式的 MCU 唤醒。经过启动时间，外加 4 个时钟周期 (此时 MCU 停止) 后， MCU 就可以运行中断服务程序了。然后 MCU 返回到 SLEEP 的下一条指令。MCU 唤醒时寄存器文件和 SRAM 的内容不会改变。如果在休眠过程中发生了复位，则 MCU 从中断向量开始执行。

P22Figure 12 介绍了 ATmega48/88/168不同的时钟系统及其分布。此图在选择合适的休眠模式时非常有用。

休眠模式控制寄存器—SMCR

休眠模式控制寄存器包含了电源管理的控制位。

位	7	6	5	4	3	2	1	0	
	–	–	–	–	SM2	SM1	SM0	SE	SMCR
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..4 Res: 保留位**

ATmega48/88/168 中的这些位都没有使用到，读返回值始终是 "0”。

- **位 3、 2、 1 – SM2..0: 休眠模式选择位 2、 1 和 0**

如 Table18 所示，这些位用于选择具体的休眠模式。

Table 18. 休眠模式选择

SM2	SM1	SM0	休眠模式
0	0	0	空闲模式
0	0	1	ADC 噪声抑制模式
0	1	0	掉电模式
0	1	1	省电模式
1	0	0	保留
1	0	1	保留
1	1	0	Standby(1) 模式
1	1	1	保留

Note: 1. 仅在使用外部晶体或谐振器时 Standby 模式才可用

- **位 0 – SE: 休眠使能**

为了使 MCU 在执行 SLEEP 指令后进入休眠模式， SE 必须置位。为了确保进入休眠模式是程序员的有意行为，建议仅在 SLEEP 指令的前一条指令置位 SE。一旦唤醒立即清除 SE。

空闲模式

SM2..0 为 000 时，SLEEP 指令使 MCU 进入空闲模式。在此模式下，CPU 停止运行，而 SPI、USART、模拟比较器、ADC、两线串行接口、定时器 / 计数器、看门狗和中断系统继续工作。这个休眠模式只停止了 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

象定时器溢出与 USART 传输完成等内外部中断都可以唤醒 MCU。如果不需要从模拟比较器中断唤醒 MCU，为了减少功耗，可以切断比较器的电源。方法是置位模拟比较器控制和状态寄存器 ACSR 的 ACD。如果 ADC 使能，进入此模式后将自动启动一次转换。

ADC 噪声抑制模式

SM2..0 为 001 时，SLEEP 指令使 MCU 进入噪声抑制模式。在此模式下，CPU 停止运行，而 ADC、外部中断、两线串行地址匹配、定时器 / 计数器 2 和看门狗继续工作 (如果已经使能)。这个休眠模式只停止了 $clk_{I/O}$、 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

此模式改善了 ADC 的噪声环境，使得转换精度更高。 ADC 使能的时候，进入此模式将自动启动一次 AD 转换。ADC 转换结束中断、外部复位、看门狗复位、 BOD 复位、两线串行地址匹配、定时器 / 计数器 2 中断、SPM/EEPROM 准备好中断、外部中断 INT0 、INT1 或引脚电平变化中断可以将 MCU 从 ADC 噪声抑制模式唤醒。

掉电模式

SM2..0 为 010 时，SLEEP 指令使 MCU 进入掉电模式。在此模式下，外部晶体停振，而外部中断、两线串行地址匹配、看门狗 (如果使能的话) 继续工作。只有外部复位、看门狗复位、看门狗中断、 BOD 复位、两线串行地址匹配、外部电平中断 INT0 或 INT1，以及引脚电平变化中断可以使 MCU 脱离掉电模式。这个休眠模式基本停止了所有的时钟，只有异步模块可以继续工作。

使用外部电平中断方式将 MCU 从掉电模式唤醒时，必须使外部电平保持一定的时间。具体请参见 P76” 外部中断 ” 。

从施加掉电唤醒条件到真正唤醒 MCU 有一个延迟时间，此时间用于时钟重新启动并稳定下来。唤醒时间与熔丝位CKSEL定义的复位时间是一样的，具体描述参见 P24”时钟源” 。

省电模式

SM2..0为011时，SLEEP指令使MCU进入省电模式。这一模式与掉电模式只有一点不同:

如果定时器 / 计数器 2 及 / 是使能的，在器件休眠期间它们继续运行。除了掉电模式的唤醒方式，定时器 / 计数器 2 的溢出中断和比较匹配中断也可以将 MCU 从休眠方式唤醒，只要 TIMSK2 使能了这些中断，而且 SREG 的全局中断使能位 I 置位。

如果定时器 / 计数器 2 无需运行，建议使用掉电模式而不是省电模式。

定时器 / 计数器 2 在省电模式下可采用同步与异步时钟驱动。如果定时器 / 计数器 2 未采用异步时钟，休眠期间定时 / 计数振荡器将停止；如果定时器 / 计数器 2 未采用同步时钟，休眠期间时钟源将停止。要注意的是，在省电模式下同步时钟只对定时器 / 计数器 2 有效。

Standby 模式

当 SM2..0 为 110 ，且选择了外部晶体振荡器或陶瓷谐振器作为时钟源， SLEEP 指令使MCU进入Standby模式。这一模式与掉电模式唯一的不同之处在于振荡器继续工作。其唤醒时间只需要 6 个时钟周期。

Table 19. 在不同睡眠模式下活动的时钟以及唤醒 MCU 的来源

	工作的时钟					振荡器		唤醒源					
睡眠模式	**clk_{CPU}**	**clk_{FLASH}**	**clk_{IO}**	**clk_{ADC}**	**clk_{ASY}**	使能的主时钟	使能的定时器时钟	**INT1, INT0 与 Pin 电平变化**	**TWI 地址匹配**	定时器 2	**SPM/ EEPROM** 准备好	**ADC**	**其它 I/O**
空闲模式			X	X	X	X	X(2)	X	X	X	X	X	X
ADC 噪声抑制模式				X	X	X	X(2)	X(3)	X	X(2)	X	X	
掉电模式								X(3)	X				
省电模式					X		X	X(3)	X	X			
Standby(1) 模式						X		X(3)	X				

Notes: 1. 时钟源为外部晶体或谐振器
2. 定时器 / 计数器 2 工作在异步模式下
3. 电平类型的 INT1 与 INT0 中断

功耗最小化

试图降低 AVR 控制系统的功耗需要考虑几个问题。一般来说，要尽可能利用休眠模式，并且使尽可能少的模块继续工作。不需要的功能必须禁止。下面的模块需要特殊考虑以达到尽可能低的功耗。

模数转换器

使能时， ADC 在所有休眠模式下都继续工作。为了降低功耗，在进入休眠模式之前需要禁止 ADC。重新启动后的第一次转换为扩展的转换。详见 P217” 模数转换器 ” 。

模拟比较器

在空闲模式时，如果没有使用模拟比较器，可以将其关闭。在 ADC 噪声抑制模式下也是如此。在其他休眠模式模拟比较器是自动关闭的。如果模拟比较器使用了内部电压基准源，则不论在什么休眠模式下都需要通过程序来关闭它。否则内部电压基准源将一直使能。请参见 P214” 模拟比较器 ” 以了解如何配置模拟比较器。

掉电检测 BOD

如果系统没有利用掉电检测器 BOD，这个模块也可以关闭。如果编程熔丝位 BODLEVEL 使能 BOD 功能，它将在各种休眠模式下继续工作，从而消耗电流。在深层次的休眠模式下，这个电流将占总电流的很大比重。请参看 P40” 掉电检测 ” 以了解如何配置 BOD。

片内基准电压

当使用 BOD、模拟比较器或 ADC 时可能需要内部电压基准源。若这些模块都禁止了，则基准源将被禁止，从而不会消耗能量。重新使能后用户必须等待基准源稳定之后才可以使用。如果基准源在休眠过程中是使能的，其输出立即可以使用。请参见 P43” 片内基准电压 ” 以了解基准源启动时间的细节。

看门狗定时器

如果系统无需利用看门狗，这个模块就可以关闭。若使能，则在任何休眠模式下都持续工作，从而消耗电流。在深层次的睡眠模式下，这个电流将占总电流的很大比重。请参看 P44” 看门狗定时器 ” 以了解如何配置看门狗定时器。

端口引脚

进入休眠模式时，所有的端口引脚都应该配置为只消耗最小的功耗。最重要的是避免驱动电阻性负载。在休眠模式下 I/O 时钟 $clk_{I/O}$ 和 ADC 时钟 clk_{ADC} 都被停止了，输入缓冲器也禁止了，从而保证输入电路不会消耗电流。在某些情况下输入逻辑是使能的，用来检测

唤醒条件。用于此功能的具体引脚请参见 P63” 数字输入使能和休眠模式 ” 。如果输入缓冲器是使能的，此时输入不能悬空，信号电平也不应该接近 $V_{CC}/2$，否则输入缓冲器会消耗额外的电流。

模拟输入引脚的数字输入缓冲器应一直禁用。否则，即使当输入引脚工作于模拟输入状态，当模拟信号电压接近 $V_{CC}/2$ 时输入缓冲器需要消耗很大的电流。可以通过操作数字输入禁止寄存器 (DIDR1 与 DIDR0) 来禁止数字输入缓冲器。具体参见 P216” 数字输入禁止寄存器 1 – DIDR1” 与 P231” 数字输入禁止寄存器 0 – DIDR0” 。

片上调试系统

如果通过熔丝位 DWEN 使能了片上调试系统，当芯片进入休眠模式时主时钟保持运行。在休眠模式中这个电流占总电流的很大比重。

复位 AVR

复位时所有的 I/O 寄存器都被设置为初始值，程序从复位向量处开始执行。对于 ATmega168 复位向量处的指令必须是绝对跳转 JMP 指令，以使程序跳转到复位处理例程。对于 ATmega48 与 ATmega88 复位向量处的指令必须是相对跳转 RJMP 指令，以使程序跳转到复位处理例程。如果程序永远不利用中断功能，中断向量可以由一般的程序代码所覆盖。这个处理方法同样适用于复位向量位于应用程序区，中断向量位于 Boot 区 — 或者反过来 — 的时候 (只适用于 ATmega88/168)。Figure16 为复位逻辑的电路图。Table20 则定义了复位电路的电气参数。

复位源有效时 I/O 端口立即复位为初始值。此时不要求任何时钟处于正常运行状态。

所有的复位信号消失之后，芯片内部的一个延迟计数器被激活，将内部复位的时间延长。这种处理方式使得在 MCU 正常工作之前有一定的时间让电源达到稳定的电平。延迟计数器的溢出时间通过熔丝位 SUT 与 CKSEL 设定。延迟时间的选择请参见 P24” 时钟源 ” 。

复位源

ATmega48/88/168 有 4 个复位源：

- 上电复位。当电源电压低于上电复位门限 V_{POT} 时， MCU 复位。
- 外部复位。当引脚 $\overline{\text{RESET}}$ 的低电平持续时间大于最小脉冲宽度时 MCU 复位。
- 看门狗复位。当看门狗使能并且看门狗定时器溢出时复位发生。
- 掉电检测复位 (BOD)。当掉电检测复位功能使能，且电源电压低于掉电检测复位门限 V_{BOT} 时 MCU 即复位。

Figure 16. 复位逻辑

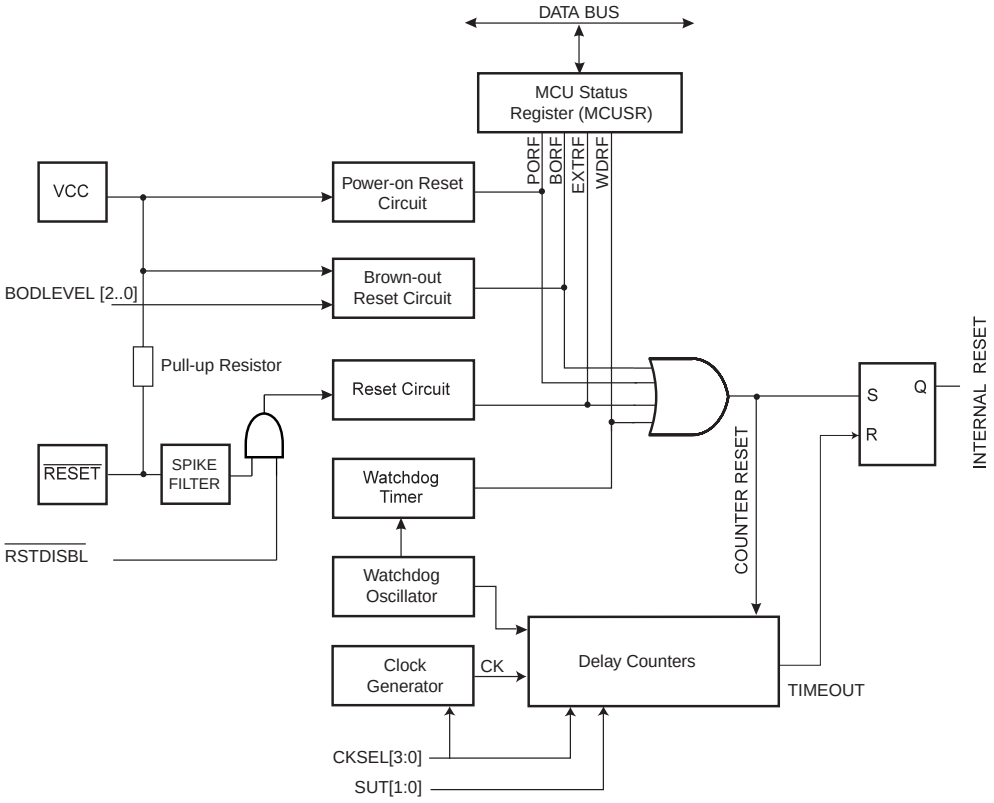


Table 20. 复位特性 (1)

符号	参数	条件	最小值	典型值	最大值	单位
V_{POT}	上电复位门限电压 (电压由低到高上升)		TBD	TBD	TBD	V
	上电复位门限电压 (电压由高到低跌落)(2)		TBD	TBD	TBD	V
V_{RST}	$\overline{\text{RESET}}$ 门限电压		0.1		0.9	V
t_{RST}	$\overline{\text{RESET}}$ 最小脉冲宽度				2.5	µs

Notes: 1. 以上数据均为指导值，实际值待测
2. 电压跌落时，只有电压低于 V_{POT} 时复位才会发生

上电复位

上电复位 (POR) 脉冲由片内检测电路产生。检测电平请参见 Table20。无论何时 V_{CC} 低于检测电平 POR 即发生。 POR 电路可以用来触发启动复位，或者用来检测电源故障。

POR 电路保证器件在上电时复位。V_{CC} 达到上电门限电压后触发延迟计数器。在计数器溢出之前器件一直保持为复位状态。当 V_{CC} 下降时，只要低于检测门限，RESET 信号立即生效。

Figure 17. MCU 启动过程，$\overline{\text{RESET}}$ 连接到 V_{CC}

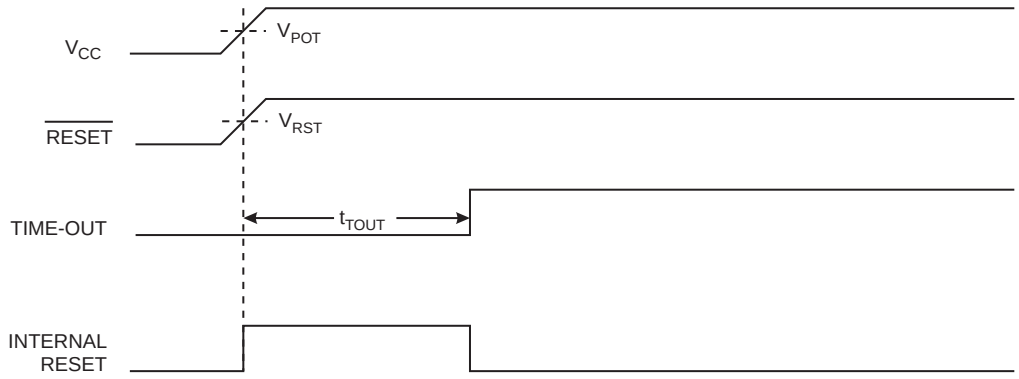
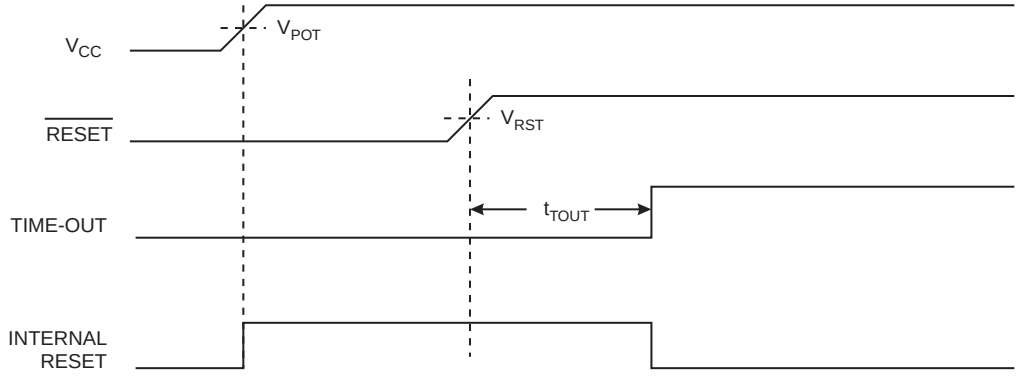


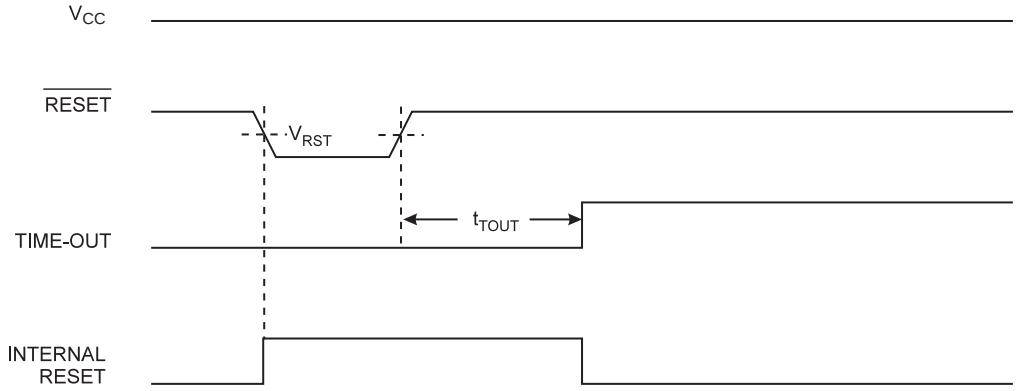
Figure 18. MCU 启动过程，$\overline{\text{RESET}}$ 由外电路控制



外部复位

外部复位由外加于 $\overline{\text{RESET}}$ 引脚的低电平产生。当复位低电平持续时间大于最小脉冲宽度时 (参见 Table20) 即触发复位过程，即使此时并没有时钟信号在运行。当外加信号达到复位门限电压 V_{RST}(上升沿) 时，t_{TOUT} 延时周期启动。延时结束后 MCU 即启动。外部复位可由 RSTDISBL 熔丝位禁用，请参见 P256Table120 。

Figure 19. 工作过程中发生外部复位



掉电检测

ATmega48/88/168 具有片内 BOD(Brown-out Detection) 电路，通过与固定的触发电平的对比来检测工作过程中 V_{CC} 的变化。此触发电平通过熔丝位 BODLEVEL 来设定。 BOD 的触发电平具有迟滞回线以消除电源尖峰的影响。这个迟滞功能可以解释为 $V_{BOT+} = V_{BOT} + V_{HYST}/2$ 以及 $V_{BOT-} = V_{BOT} - V_{HYST}/2$。

Table 21. BODLEVEL 熔丝位编码 (1)

BODLEVEL 2..0 熔丝位	最小 V_{BOT}	典型 V_{BOT}	最大 V_{BOT}	单位
111	BOD 被禁用			
110		1.8		V
101		2.7		
100		4.3		
011	保留			
010				
001				
000				

Note: 1. V_{BOT} 可能低于某些器件的最小标称工作电压。对于有这种情形的器件，在产品测试时将做 $V_{CC} = V_{BOT}$ 的实验。这保证了在芯片工作电压 V_{CC} 降至微处理器已经无法正常工作之前，掉电复位必定发生。ATmega48V/88V/168V 用 BODLEVEL = 110 与 BODLEVEL = 101 做检测，ATmega48/88/168 用 BODLEVEL = 101 与 BODLEVEL = 101 做检测。

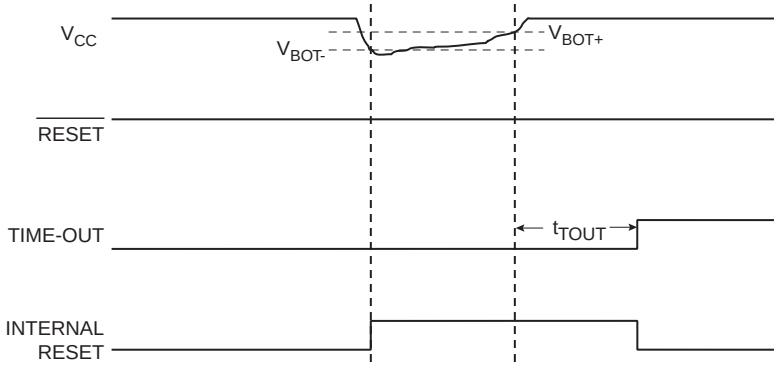
Table 22. 掉电检测特性

符号	参数	最小值	典型值	最大值	单位
V_{HYST}	掉电检测迟滞		50		mV
t_{BOD}	掉电复位最小脉宽				ns

BOD 使能后，一旦 V_{CC} 下降到触发电平以下 (V_{BOT-} ，Figure20)，BOD 复位立即被激发。当 V_{CC} 上升到触发电平以上时 (V_{BOT+}，Figure20)，延时计数器开始计数，一旦超过溢出时间 t_{TOUT}， MCU 即恢复工作。

如果 V_{CC} 一直低于触发电平并保持如 Table20 所示的时间 t_{BOD}，BOD 电路将只检测电压跌落。

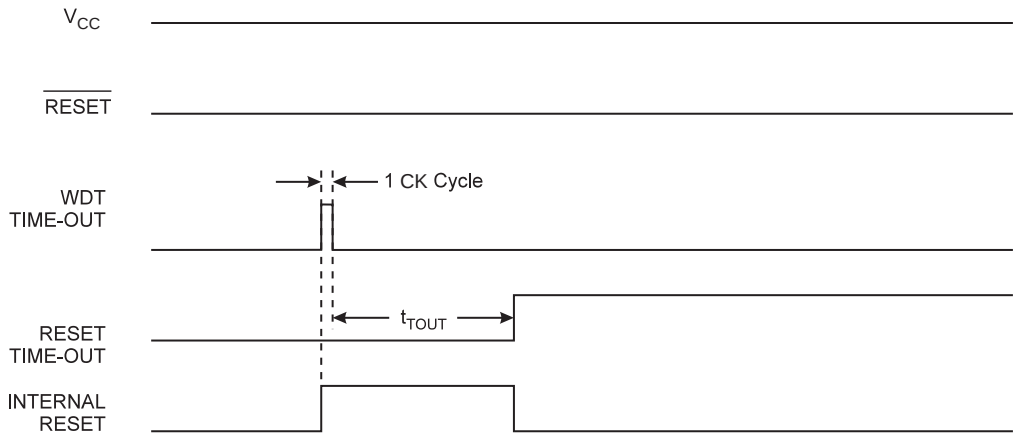
Figure 20. 工作过程中发生掉电检测复位



看门狗复位

看门狗定时器溢出时将产生持续时间为 1 个 CK 周期的复位脉冲。在脉冲的下降沿，延时定时器开始对 t_{TOUT} 记数。请参见 P44 以了解看门狗定时器的具体操作过程。

Figure 21. 工作过程中发生看门狗复位



MCU 状态寄存器—MCUSR

MCU 状态寄存器提供了有关引起 MCU 复位的复位源的信息。

位	7	6	5	4	3	2	1	0	
	–	–	–	–	WDRF	BORF	EXTRF	PORF	MCUSR
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	参见各个位的说明				

- **位 7..4: Res: 保留位**

ATmega48/88/168 中的这些位都没有适用，读返回值始终为 "0”。

- **位 3 – WDRF: 看门狗复位标志**

看门狗复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **位 2 – BORF: 掉电检测复位标志**

掉电检测复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **位 1 – EXTRF: 外部复位标志**

外部复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **位 0 – PORF: 上电复位标志**

上电复位发生时置位。只能通过写 ”0” 来清除。

为了使用这些复位标志来识别复位条件，用户应该尽早读取 MCUSR 的数据，然后将其复位。如果在其他复位发生之前将此寄存器复位，则后续复位源可以通过检查复位标志来识别。

片内基准电压

ATmega48/88/168 具有片内能隙基准源，用于掉电检测，或者是作为模拟比较器或 ADC 的输入。

基准电压使能信号和启动时间

电压基准的启动时间可能影响其工作方式。启动时间列于 Table23。为了降低功耗，可以控制基准源仅在如下情况打开：

1. BOD 使能 (熔丝位 BODLEVEL [2..0] 被编程)
2. 能隙基准源连接到模拟比较器 (ACSR 寄存器的 ACBG 置位)
3. ADC 使能

因此，当 BOD 被禁止时，置位 ACBG 或使能 ADC 后要等待基准源启动之后才能使用这些功能。为了降低掉电模式的功耗，在进入掉电模式之前用户可以禁止上述三种条件以关闭基准源。

Table 23. 内部电压基准源的特性 (1)

符号	参数	条件	最小值	典型值	最大值	单位
V_{BG}	能隙基准源电压	TBD	1.0	1.1	1.2	V
t_{BG}	能隙基准源启动时间	TBD		40	70	µs
I_{BG}	能隙基准源功耗	TBD		10	TBD	µA

Note: 1. 上述只是参考值，实际值待定

看门狗定时器

看门狗定时器由独立的 128 kHz 片内振荡器驱动。通过设置看门狗定时器的预分频器可以调节看门狗复位的时间间隔，如 P44Table26 所示。看门狗复位指令 WDR 用来复位看门狗定时器。此外，禁止看门狗定时器或发生复位时它也被复位。复位时间有 10 个选项。如果没有及时复位定时器，一旦时间超过复位周期，ATmega48/88/168 就复位，并执行复位向量指向的程序。具体的看门狗复位时序在 P44Table26 有说明。

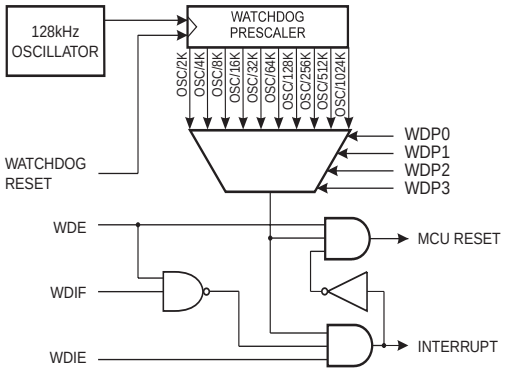
看门狗定时器还可用来产生中断。这在使用看门狗将系统从掉电状态唤醒是非常有用的。

为了防止无意之间禁止看门狗定时器或改变了复位时间，熔丝位 WDTON 为此提供了 2 个不同的保护级别，如 Table 24. 所示。请参见 P47”改变看门狗定时器配置的时间序列”。

Table 24. 由 WDTON 熔丝位设定的 WDT 配置表

WDTON	安全等级	WDT 初始状态	如何禁止 WDT	如何改变复位间隔时间
未编程	1	禁止	时间序列	
已编程	2	使能	总是使能	时间序列

Figure 22. 看门狗定时器



看门狗定时器控制寄存器—WDTCSR

位	7	6	5	4	3	2	1	0	
	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	X	0	0	0	

- **位 7 – WDIF: 看门狗超时中断标志**

当看门狗定时器超时且定时器作为中断使用时，该位置位。执行相应的中断处理程序时 WDIF 由硬件清零。也可通过对标志位写 "1” 对 WDIF 清零。当 SREG 寄存器中的 I 位与 WDIE 也置位时， MCU 执行看门狗超时中断。

- **位 6 – WDIE: 看门狗超时中断使能**

WDIE 置 "1” 时 WDE 被清零，状态寄存器中的 I 位置位，看门狗超时中断使能。当看门狗定时器出现超时时执行相应的中断程序。

如果 WDE 置位，当超时出现时，WDIE 由硬件自动清零。这对使用中断时保证看门狗复位的安全性非常有效。在 WDIE 位被清零后，下一个超时将引发系统复位。为避免看门狗复位，在每次中断后必须对 WDIE 置位。

Table 25. 看门狗定时器的配置

WDE	WDIE	看门狗定时器状态	超时后的动作
0	0	停止	无
0	1	运行	中断
1	0	运行	复位
1	1	运行	中断

- **Bit 4 – WDCE: 看门狗修改使能**

清零 WDE 时必须置位 WDCE，否则不能禁止看门狗。一旦置位，硬件将在紧接的 4 个时钟周期之后将其清零。请参考有关 WDE 的说明来禁止看门狗。修改预分频器也必须置位 WDCE，如 P47“ 改变看门狗定时器配置的时间序列 ” 所示。

- **位 3 – WDE: 使能看门狗**

WDE为"1“时，看门狗使能，否则看门狗将被禁止。只有在WDCE为"1“时WDE才能清零。以下为关闭看门狗的步骤：

1. 在同一个指令内对 WDCE 和 WDE 写 "1“，即使 WDE 已经为 "1“。
2. 在紧接的 4 个时钟周期之内对 WDE 写 "0”。这会禁用看门狗。

工作于安全级别 2 时，即使使用了上述的算法，也无法禁止看门狗定时器。参见 P47“ 改变看门狗定时器配置的时间序列 ” 所示。

工作于安全级别 1 时，WDE 被 MCUSR 的 WDRF 功能所替换 ，请参见 P41”MCU 状态寄存器 – MCUSR” 中关于 WDRF 的说明。这意味着当 WDRF 置位时 WDE 同时置位。为清零 WDE，在使用上述过程禁用看门狗之前必须清零 WDRF。这一特性保证在出现故障时有多重复位，且在故障解决后可以安全地启动。

Note: 如果在应用中不需要使用看门狗定时器，则在器件初始化时应运行看门狗禁用程序。如果看门狗被意外使能，如程序跑飞或出现 BOD，器件将会复位，且在结束复位时 WDRF 标志位置位。这将自动激活看门狗，引发新的看门狗复位。为避免出现这种状况，在初始化过程中应用程序应将 WDRF 标志位与 WDE 控制位清零 。

- **位 5, 2..0 – WDP3..0: 看门狗定时器预分频器 3, 2, 1, 与 0**

当看门狗定时器使能时， WDP3..0 决定看门狗定时器的预分频器，如 Table26 所示。

Table 26. 看门狗定时器预分频器选项

WDP3	WDP2	WDP1	WDP0	看门狗振荡器周期数	V_{CC} = 5.0V 时的典型溢出时间
0	0	0	0	2K	16 ms
0	0	0	1	4K	32 ms
0	0	1	0	8K	64 ms
0	0	1	1	16K	0.125 s
0	1	0	0	32K	0.25 s
0	1	0	1	64K	0.5 s
0	1	1	0	128K	1.0 s

Table 26. 看门狗定时器预分频器选项

WDP3	WDP2	WDP1	WDP0	看门狗振荡器周期数	V_{CC} = 5.0V 时的典型溢出时间
0	1	1	1	256K	2.0 s
1	0	0	0	512K	4.0 s
1	0	0	1	1024K	8.0 s
1	0	1	0	保留	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

下面的例子分别用汇编和 C 实现了关闭 WDT 的操作。在此假定中断处于用户控制之下 (比如已经禁止了全局中断) ，因而在执行下面程序时中断不会发生。

汇编代码例程 (1)

```
WDT_off:
  ; MCUSR 中的 WDRF 清零
  ldi   r16, (0<<WDRF)
  out   MCUSR, r16
  ; 置位 WDCE 与 WDE
  ldi  r16, (1<<WDCE)|(1<<WDE)
  out  WDTCSR, r16
  ; 关闭 WDT
  ldi  r16, (0<<WDE)
  out  WDTCSR, r16
  ret
```

C 代码例程 (1)

```
void WDT_off(void)
{
  /*MCUSR 中的 WDRF 清零*/
  MCUSR = 0x00
  /* 置位 WDCE 与 WDE */
  WDTCSR = (1<<WDCE) | (1<<WDE);
  /* 关闭 WDT */
  WDTCSR = 0x00;
}
```

Note: 1. 例程假定已经包含了所有必须的头文件

改变看门狗定时器配置的时间序列

改变配置的序列根据不同的安全级别略有不同。下面将分别加以描述：

安全级别 1

在此模式下看门狗定时器的初始状态是禁止的，可以通过置位 WDE 来使能它。改变定时器溢出时间及禁止 (已经使能的) 看门狗定时器需要执行一个特定的时间序列：

1. 在同一个指令内对 WDCE 和 WDE 写 "1“，即使 WDE 已经为 "1“。
2. 在紧接的 4 个时钟周期之内将 WDE 和 WDP 设置为合适的值，而 WDCE 写 "0”。

安全级别 2

在此模式下看门狗定时器总是使能的， WDE 的读返回值总是为 "1”。改变定时器溢出时间需要执行一个特定的时间序列：

1. 在同一个指令内对WDCE和WDE写"1“。虽然WDE总是为置位状态，也必须写"1“以启动时序。
2. 在紧接的 4 个时钟周期之内同时对 WDCE 写 "0”，以及为 WDP 写入合适的数据。WDE 的数值可以任意。

中断

本节描述 ATmega48/88/168 的中断处理。更一般的 AVR 中断处理请参见 P11” 复位与中断处理 ”。

ATmega48、ATmega88 与 ATmega168 的中断向量基本相同，只有如下差别：

- ATmega168 的每个中断向量占据两个指令字，而 ATmega48 与 ATmega88 的只占一个指令字。
- ATmega48 没有独立的 Boot Loader 区。在 ATmega88 与 ATmega168 中，复位向量由 BOOTRST 熔丝位决定，中断向量的起始地址由 MCUCR 寄存器的 IVSEL 决定。

ATmega48 的中断向量

Table 27. ATmega48 的复位和中断向量

向量号	程序地址 (2)	中断源	中断定义
1	0x000	RESET	外部电平复位，上电复位，掉电检测复位，看门狗复位
2	0x001	INT0	外部中断请求 0
3	0x002	INT1	外部中断请求 1
4	0x003	PCINT0	引脚电平变化中断请求 0
5	0x004	PCINT1	引脚电平变化中断请求 1
6	0x005	PCINT2	引脚电平变化中断请求 2
7	0x006	WDT	看门狗超时中断
8	0x007	TIMER2 COMPA	定时器 / 计数器 2 比较匹配 A
9	0x008	TIMER2 COMPB	定时器 / 计数器 2 比较匹配 B
10	0x009	TIMER2 OVF	定时器 / 计数器 2 溢出
11	0x00A	TIMER1 CAPT	定时器 / 计数器 1 事件捕捉
12	0x00B	TIMER1 COMPA	定时器 / 计数器 1 比较匹配 A
13	0x00C	TIMER1 COMPB	定时器 / 计数器 1 比较匹配 B
14	0x00D	TIMER1 OVF	定时器 / 计数器 1 溢出
15	0x00E	TIMER0 COMPA	定时器 / 计数器 0 比较匹配 A
16	0x00F	TIMER0 COMPB	定时器 / 计数器 0 比较匹配 B
17	0x010	TIMER0 OVF	定时器 / 计数器 0 溢出
18	0x011	SPI, STC	SPI 串行传输结束
19	0x012	USART, RX	USART, Rx 结束
20	0x013	USART, UDRE	USART 数据寄存器空
21	0x014	USART, TX	USART, Tx 结束
22	0x015	ADC	ADC 转换结束
23	0x016	EE READY	EEPROM 准备好
24	0x017	ANALOG COMP	模拟比较器
25	0x018	TWI	两线串行接口
26	0x019	SPM READY	保存程序存储器内容就绪

ATmega48 典型的复位与中断向量设置如下：

```
地址      标号     代码                                说明
0x000             rjmp    RESET              ; 复位处理
0x001             rjmp    EXT_INT0           ; IRQ0 处理
0x002             rjmp    EXT_INT1           ; IRQ1 处理
0x003             rjmp    PCINT0             ; PCINT0 处理
0x004             rjmp    PCINT1             ; PCINT1 处理
0x005             rjmp    PCINT2             ; PCINT2 处理
0x006             rjmp    WDT                ; 看门狗定时器处理
0x007             rjmp    TIM2_COMPA         ; 定时器 2 比较 A 处理
0x008             rjmp    TIM2_COMPB         ; 定时器 2 比较 B 处理
0x009             rjmp    TIM2_OVF           ; 定时器 2 溢出处理
0x00A             rjmp    TIM1_CAPT          ; 定时器 1 捕获处理
0x00B             rjmp    TIM1_COMPA         ; 定时器 1 比较 A 处理
0x00C             rjmp    TIM1_COMPB         ; 定时器 1 比较 B 处理
0x00D             rjmp    TIM1_OVF           ; 定时器 1 溢出处理
0x00E             rjmp    TIM0_COMPA         ; 定时器 0 比较 A 处理
0x00F             rjmp    TIM0_COMPB         ; 定时器 0 比较 B 处理
0x010             rjmp    TIM0_OVF           ; 定时器 0 溢出处理
0x011             rjmp    SPI_STC            ; SPI 传输结束处理
0x012             rjmp    USART_RXC          ; USART，RX 结束处理
0x013             rjmp    USART_UDRE         ; USART，UDR 空处理
0x014             rjmp    USART_TXC          ; USART，TX Complete 处理
0x015             rjmp    ADC                ; ADC 转换结束处理
0x016             rjmp    EE_RDY             ; EEPROM 就绪处理
0x017             rjmp    ANA_COMP           ; 模拟比较器处理
0x018             rjmp    TWI                ; 两线串行接口处理
0x019             rjmp    SPM_RDY            ; SPM 就绪处理
;
0x01A RESET:      ldi     r16, high(RAMEND); 主程序
0x01B             out     SPH,r16            ; 设置堆栈指针为 RAM 的顶部
0x01C             ldi     r16, low(RAMEND)
0x01D             out     SPL,r16
0x01E             sei                        ; 使能中断
0x01F             <instr>  xxx
  ...      ...     ...    ...
```

ATmega88 的中断向量

Table 28. ATmega88 的复位和中断向量

向量号	程序地址 (2)	中断源	中断定义
1	0x000(1)	RESET	外部电平复位，上电复位，掉电检测复位，看门狗复位
2	0x001	INT0	外部中断请求 0
3	0x002	INT1	外部中断请求 1
4	0x003	PCINT0	引脚电平变化中断请求 0
5	0x004	PCINT1	引脚电平变化中断请求 1
6	0x005	PCINT2	引脚电平变化中断请求 2
7	0x006	WDT	看门狗超时中断
8	0x007	TIMER2 COMPA	定时器 / 计数器 2 比较匹配 A
9	0x008	TIMER2 COMPB	定时器 / 计数器 2 比较匹配 B
10	0x009	TIMER2 OVF	定时器 / 计数器 2 溢出
11	0x00A	TIMER1 CAPT	定时器 / 计数器 1 事件捕捉
12	0x00B	TIMER1 COMPA	定时器 / 计数器 1 比较匹配 A
13	0x00C	TIMER1 COMPB	定时器 / 计数器 1 比较匹配 B
14	0x00D	TIMER1 OVF	定时器 / 计数器 1 溢出
15	0x00E	TIMER0 COMPA	定时器 / 计数器 0 比较匹配 A
16	0x00F	TIMER0 COMPB	定时器 / 计数器 0 比较匹配 B
17	0x010	TIMER0 OVF	定时器 / 计数器 0 溢出
18	0x011	SPI, STC	SPI 串行传输结束
19	0x012	USART, RX	USART, Rx 结束
20	0x013	USART, UDRE	USART 数据寄存器空
21	0x014	USART, TX	USART, Tx Complete
22	0x015	ADC	ADC 转换结束
23	0x016	EE READY	EEPROM 准备好
24	0x017	ANALOG COMP	模拟比较器
25	0x018	TWI	两线串行接口
26	0x019	SPM READY	保存程序存储器内容就绪

Notes: 1. 熔丝位BOOTRST被编程时，MCU复位后程序跳转到Boot Loader。请参见 P241”Boot Loader 支持 RWW 自编程， ATmega88 与 ATmega168”

2. 当寄存器MCUCR的IVSEL置位时，中断向量转移到Boot区的起始地址。此时各个中断向量的实际地址为表中地址与 Boot 区起始地址之和。

Table29 给出了不同的BOOTRST/IVSEL设置条件下的复位和中断向量的位置。如果程序没有使用中断，中断向量就没有意义。用户可以在此直接写程序。同样，如果复位向量位于应用区，而其他中断向量位于 Boot 区，则复位向量之后可以直接写程序。反过来亦是如此。

Table 29. ATmega88[1] 复位和中断向量位置的确定

BOOTRST	IVSEL	复位地址	中断向量起始地址
1	0	0x000	0x001
1	1	0x000	Boot 区复位地址 + 0x001
0	0	Boot 区复位地址	0x001
0	1	Boot 区复位地址	Boot 区复位地址 + 0x001

Note: 1. Boot区复位地址列于P252Table109 。对于熔丝位BOOTRST ,“1”表示未编程，“0”表示已编程

ATmega88 典型的复位和中断设置如下：

```
地址     标号     代码                          说明
0x000            rjmp    RESET            ; 复位处理
0x001            rjmp    EXT_INT0         ; IRQ0 处理
0x002            rjmp    EXT_INT1         ; IRQ1 处理
0x003            rjmp    PCINT0           ; PCINT0 处理
0x004            rjmp    PCINT1           ; PCINT1 处理
0x005            rjmp    PCINT2           ; PCINT2 处理
0x006            rjmp    WDT              ; 看门狗定时器处理
0x007            rjmp    TIM2_COMPA       ; 定时器 2 比较 A 处理
0X008            rjmp    TIM2_COMPB       ; 定时器 2 比较 B 处理
0x009            rjmp    TIM2_OVF         ; 定时器 2 溢出处理
0x00A            rjmp    TIM1_CAPT        ; 定时器 1 捕捉处理
0x00B            rjmp    TIM1_COMPA       ; 定时器 1 比较 A 处理
0x00C            rjmp    TIM1_COMPB       ; 定时器 1 比较 B 处理
0x00D            rjmp    TIM1_OVF         ; 定时器 1 比较处理
0x00E            rjmp    TIM0_COMPA       ; 定时器 0 比较 A 处理
0x00F            rjmp    TIM0_COMPB       ; 定时器 0 比较 B 处理
0x010            rjmp    TIM0_OVF         ; 定时器 0 溢出处理
0x011            rjmp    SPI_STC          ; SPI 传输结束处理
0x012            rjmp    USART_RXC        ; USART， RX 结束处理
0x013            rjmp    USART_UDRE       ; USART， UDR 空处理
0x014            rjmp    USART_TXC        ; USART， TX 结束处理
0x015            rjmp    ADC              ; ADC 转换结束处理
0x016            rjmp    EE_RDY           ; EEPROM 就绪处理
0x017            rjmp    ANA_COMP         ; 模拟比较器处理
0x018            rjmp    TWI              ; 两线串行接口处理
0x019            rjmp    SPM_RDY          ; SPM 就绪处理
;
0x01A RESET:     ldi     r16, high(RAMEND); 主程序
0x01B            out     SPH,r16          ; 设置堆栈指针为 RAM 的顶部
0x01C            ldi     r16, low(RAMEND)
0x01D            out     SPL,r16
0x01E            sei                      ; 使能中断
0x01F            <instr>  xxx
  ...      ...    ...    ...
```

当熔丝位 BOOTRST 未编程， Boot 区为 2K 字节，且寄存器 MCUCR 的 IVSEL 在使能任何中断之前得到置位时， ATmega88 典型的复位和中断设置如下：

```
地址      标号     代码                    说明
0x000     RESET:  ldi     r16,high(RAMEND) ; 主程序
0x001             out     SPH,r16          ; 设置堆栈指针为 RAM 的顶部
0x002             ldi     r16,low(RAMEND)
0x003             out     SPL,r16
0x004             sei                      ; 使能中断
0x005             <instr>  xxx
;
.org 0xC01
0xC01             rjmp    EXT_INT0         ; IRQ0 处理
0xC02             rjmp    EXT_INT1         ; IRQ1 处理
...               ...     ...              ;
0xC19             rjmp    SPM_RDY          ; SPM 就绪处理
```

当熔丝位 BOOTRST 被编程，且 Boot 区为 2K 字节时，ATmega88 典型的复位和中断设置如下：

```
地址      标号代码          解释
.org 0x001
0x001             rjmp    EXT_INT0         ; IRQ0 处理
0x002             rjmp    EXT_INT1         ; IRQ1 处理
...               ...     ...              ;
0x019             rjmp    SPM_RDY          ; SPM 就绪处理
;
.org 0xC00
0xC00     RESET:  ldi     r16,high(RAMEND) ; 主程序
0xC01             out     SPH,r16          ; 设置堆栈指针为 RAM 的顶部
0xC02             ldi     r16,low(RAMEND)
0xC03             out     SPL,r16
0xC04             sei                      ; 使能中断
0xC05             <instr>  xxx
```

当熔丝位 BOOTRST 被编程，Boot 区为 2K 字节，且寄存器 MCUCR 的 IVSEL 在使能任何中断之前被置位时， ATmega88 典型的复位和中断设置如下：

```
地址      标号     代码                    说明
;
.org 0xC00
0xC00             rjmp    RESET           ; 复位处理
0xC01             rjmp    EXT_INT0        ; IRQ0 处理
0xC02             rjmp    EXT_INT1        ; IRQ1 处理
...               ...     ...             ;
0xC19             rjmp    SPM_RDY         ; SPM 就绪处理
;
0xC1A     RESET:  ldi     r16,high(RAMEND) ; 主程序
0xC1B             out     SPH,r16         ; 设置堆栈指针为 RAM 的顶部
0xC1C             ldi     r16,low(RAMEND)
0xC1D             out     SPL,r16
0xC1E             sei                     ; 使能中断
0xC1F             <instr>  xxx
```

ATmega168 的中断向量

Table 30. ATmega168 的复位和中断向量

向量号	程序地址 (2)	中断源	中断定义
1	0x0000(1)	RESET	外部电平复位，上电复位，掉电检测复位，看门狗复位
2	0x0002	INT0	外部中断请求 0
3	0x0004	INT1	外部中断请求 1
4	0x0006	PCINT0	引脚电平变化中断请求 0
5	0x0008	PCINT1	引脚电平变化中断请求 1
6	0x000A	PCINT2	引脚电平变化中断请求 2
7	0x000C	WDT	看门狗超时中断
8	0x000E	TIMER2 COMPA	定时器 / 计数器 2 比较匹配 A
9	0x0010	TIMER2 COMPB	定时器 / 计数器 2 比较匹配 B
10	0x0012	TIMER2 OVF	定时器 / 计数器 2 溢出
11	0x0014	TIMER1 CAPT	定时器 / 计数器 1 事件捕捉
12	0x0016	TIMER1 COMPA	定时器 / 计数器 1 比较匹配 A
13	0x0018	TIMER1 COMPB	定时器 / 计数器 1 比较匹配 B
14	0x001A	TIMER1 OVF	定时器 / 计数器 1 溢出
15	0x001C	TIMER0 COMPA	定时器 / 计数器 0 比较匹配 A
16	0x001E	TIMER0 COMPB	定时器 / 计数器 0 比较匹配 B
17	0x0020	TIMER0 OVF	定时器 / 计数器 0 溢出
18	0x0022	SPI, STC	SPI 串行传输结束
19	0x0024	USART, RX	USART Rx 结束
20	0x0026	USART, UDRE	USART，数据寄存器空
21	0x0028	USART, TX	USART， Tx 结束

Table 30. ATmega168 的复位和中断向量 (Continued)

向量号	程序地址 (2)	中断源	中断定义
22	0x002A	ADC	ADC 转换结束
23	0x002C	EE READY	EEPROM 就绪
24	0x002E	ANALOG COMP	模拟比较器
25	0x0030	TWI	两线串行接口
26	0x0032	SPM READY	SPM 就绪

Notes: 1. 熔丝位BOOTRST被编程时，MCU复位后程序跳转到Boot Loader。请参见P241”Boot Loader 支持 RWW 自编程， ATmega88 与 ATmega168” 。
2. 当寄存器MCUCR的IVSEL置位时，中断向量转移到Boot区的起始地址。此时各个中断向量的实际地址为表中地址与 Boot 区起始地址之和。

Table31给出了不同的BOOTRST/IVSEL设置条件下的复位和中断向量的位置。如果程序永远不使用中断，中断向量就没有意义。用户可以在此直接写程序。同样，如果复位向量位于应用区，而其他中断向量位于 Boot 区，则复位向量之后可以直接写程序。反过来亦是如此。

Table 31. ATmega168(1) 复位和中断向量位置的确定

BOOTRST	IVSEL	复位地址	中断向量起始地址
1	0	0x000	0x001
1	1	0x000	Boot 区复位地址 + 0x0002
0	0	Boot 区复位地址	0x001
0	1	Boot 区复位地址	Boot 区复位地址 + 0x0002

Note: 1. Boot区复位地址列于P252Table109 。对于熔丝位BOOTRST，“1”表示未编程，“0”表示已编程。

ATmega168 典型的复位和中断设置如下：

```
地址              符号    代码              说明
0x0000            jmp     RESET             ; 复位处理
0x0002            jmp     EXT_INT0          ; IRQ0 处理
0x0004            jmp     EXT_INT1          ; IRQ1 处理
0x0006            jmp     PCINT0            ; PCINT0 处理
0x0008            jmp     PCINT1            ; PCINT1 处理
0x000A            jmp     PCINT2            ; PCINT2 处理
0x000C            jmp     WDT               ; 看门狗定时器处理
0x000E            jmp     TIM2_COMPA        ; 定时器 2 比较 A 处理
0x0010            jmp     TIM2_COMPB        ; 定时器 2 比较 B 处理
0x0012            jmp     TIM2_OVF          ; 定时器 2 溢出处理
0x0014            jmp     TIM1_CAPT         ; 定时器 1 捕捉处理
0x0016            jmp     TIM1_COMPA        ; 定时器 1 比较 A 处理
0x0018            jmp     TIM1_COMPB        ; 定时器 1 比较 B 处理
0x001A            jmp     TIM1_OVF          ; 定时器 1 溢出处理
0x001C            jmp     TIM0_COMPA        ; 定时器 0 比较 A 处理
0x001E            jmp     TIM0_COMPB        ; 定时器 0 比较 B 处理
0x0020            jmp     TIM0_OVF          ; 定时器 0 溢出处理
0x0022            jmp     SPI_STC           ; SPI 传输结束处理
0x0024            jmp     USART_RXC         ; USART， RX 结束处理
```

```
0x0026          jmp     USART_UDRE        ; USART，UDR 空处理
0x0028          jmp     USART_TXC         ; USART，TX 结束处理
0x002A          jmp     ADC               ; ADC 转换结束处理
0x002C          jmp     EE_RDY            ; EEPROM 就绪处理
0x002E          jmp     ANA_COMP          ; 模拟比较器处理
0x0030          jmp     TWI               ; 两线串行处理
0x0032          jmp     SPM_RDY           ; SPM 就绪处理
;
0x0033RESET:    ldi     r16, high(RAMEND); 主程序
0x0034          out     SPH,r16           ; 设置堆栈指针为 RAM 的顶部
0x0035          ldi     r16, low(RAMEND)
0x0036          out     SPL,r16
0x0037          sei                       ; 使能中断
0x0038          <instr>  xxx
  ...     ...    ...   ...
```

当熔丝位 BOOTRST 被编程，Boot 区为 2K 字节，且寄存器 MCUCR 的 IVSEL 在使能任何中断之前被置位时， ATmega168 典型的复位和中断设置如下：

```
地址            符号    代码                说明
0x0000   RESET: ldi     r16,high(RAMEND) ; 主程序
0x0001          out     SPH,r16          ; 设置堆栈指针为 RAM 的顶部
0x0002          ldi     r16,low(RAMEND)
0x0003          out     SPL,r16
0x0004          sei                      ; 使能中断
0x0005          <instr>  xxx
;
.org 0xC02
0x1C02          jmp     EXT_INT0         ; IRQ0 处理
0x1C04          jmp     EXT_INT1         ; IRQ1 处理
...             ...     ...              ;
0x1C32          jmp     SPM_RDY          ; SPM 就绪处理
```

当熔丝位 BOOTRST 未编程， Boot 区为 2K 字节， ATmega168 典型的复位和中断设置如下：

```
地址              符号     代码                说明
.org 0x0002
0x0002            jmp      EXT_INT0          ; IRQ0 处理
0x0004            jmp      EXT_INT1          ; IRQ1 处理
...               ...      ...               ;
0x0032            jmp      SPM_RDY           ; SPM 就绪处理
;
.org 0x1C00
0x1C00    RESET:  ldi      r16,high(RAMEND) ; 主程序
0x1C01            out      SPH,r16           ; 设置堆栈指针为 RAM 的顶部
0x1C02            ldi      r16,low(RAMEND)
0x1C03            out      SPL,r16
0x1C04            sei                        ; 使能中断
0x1C05            <instr>  xxx
```

ATmega168 当熔丝位BOOTRST被编程 ,Boot区为2K 字节 ,且寄存器MCUCR的IVSEL在使能任何中断之前被置位时， ATmega168 典型的复位和中断设置如下：

```
地址              符号     代码                说明
;
.org 0x1C00
0x1C00            jmp      RESET             ; 复位处理
0x1C02            jmp      EXT_INT0          ; IRQ0 处理
0x1C04            jmp      EXT_INT1          ; IRQ1 处理
...               ...      ...               ;
0x1C32            jmp      SPM_RDY           ; SPM 就绪处理
;
0x1C33    RESET:  ldi      r16,high(RAMEND) ; 主程序
0x1C34            out      SPH,r16           ; 设置堆栈指针为 RAM 的顶部
0x1C35            ldi      r16,low(RAMEND)
0x1C36            out      SPL,r16
0x1C37            sei                        ; 使能中断
0x1C38            <instr>  xxx
```

在应用区和 Boot 区之间移动中断，ATmega88 与 ATmega168

通用中断控制寄存器决定中断向量表的放置地址。

MCU 控制寄存器—MCUCR

位	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
读 / 写	R/W	R	R	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – IVSEL: 中断向量选择**

IVSEL为"0“时 ,中断向量位于Flash存储器的起始地址；IVSEL为"1“时 ,中断向量转移到 Boot 区的起始地址。实际的 Boot 区起始地址由熔丝位 BOOTSZ 确定。具体请参考 P241”Boot Loader 支持 RWW 自编程 ,ATmega88 与 ATmega168” 。为了防止无意识地改变中断向量表，修改 IVSEL 时需要遵循如下过程：

1. 置位中断向量修改使能位 IVCE
2. 在紧接的 4 个时钟周期里将需要的数据写入 IVSEL，同时对 IVCE 写 ”0”

执行上述序列时中断自动被禁止。其实，在置位 IVCE 时中断就被禁止了，并一直保持到写 IVSEL 操作之后的下一条语句。如果没有 IVSEL 写操作，则中断在置位 IVCE 之后的 4 个时钟周期保持禁止。需要注意的是，虽然中断被自动禁止，但状态寄存器的位 I 并不会因此而受到影响。

Note: 若中断向量位于Boot Loader区，且Boot锁定位BLB02被编程，则执行应用区的程序时中断被禁止；若中断向量位于应用区，且 Boot 锁定位 BLB12 被编程，则执行 Boot Loader 区的程序时中断被禁止。有关 Boot 锁定位的细节请参见 P241”Boot Loader 支持 RWW 自编程， ATmega88 与 ATmega168”。

该位在 ATmega48 中无效。

- **位 0 – IVCE: 中断向量修改使能**

改变 IVSEL 时 IVCE 必须置位。在 IVCE 写入 4 个时钟周期或 IVSEL 写操作之后，IVCE 被硬件清零。如前面所述，置位 IVCE 将禁止中断。代码如下：

汇编代码例程：

```
Move_interrupts:
  ; 使能中断向量的修改
  ldi  r16, (1<<IVCE)
  out  MCUCR, r16
  ; 将中断向量转移到 boot 区
  ldi  r16, (1<<IVSEL)
  out  MCUCR, r16
  ret
```

C 代码例程

```
void Move_interrupts(void)
{
  /* 使能中断向量的修改 */
  MCUCR = (1<<IVCE);
  /* 将中断向量转移到 boot 区 */
  MCUCR = (1<<IVSEL);
}
```

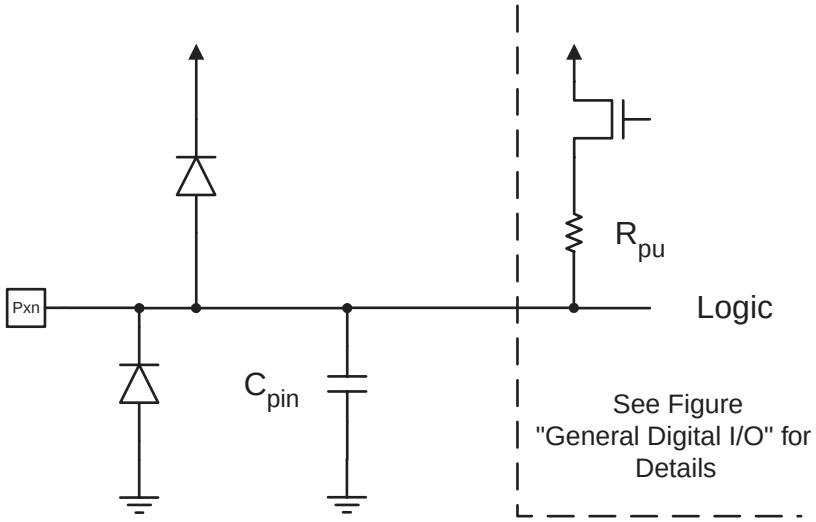
该位在 ATmega48 中无效。

I/O 端口

介绍

作为通用数字 I/O 使用时，AVR 所有的 I/O 端口都具有真正的读 - 修改 - 写功能。这意味着用 SBI 或 CBI 指令改变某些管脚的方向 (或者是端口电平、禁止 / 使能上拉电阻) 时不会改变其他管脚的方向 (或者是端口电平、禁止 / 使能上拉电阻)。输出缓冲器具有对称的驱动能力，可以输出或吸收大电流，直接驱动 LED。所有的端口引脚都具有与电压无关的上拉电阻。并有保护二极管与 V_{CC} 和地相连，如 Figure23 所示。请参见 P275” 电气特性 ” 以了解完整的参数。

Figure 23. I/O 引脚等效原理图



本节所有的寄存器和位以通用格式表示 : 小写的 “x” 表示端口的序号，而小写的 “n” 代表位的序号。但是在程序里要写完整。例如，PORTB3 表示端口 B 的第 3 位，而本节的通用格式为 PORTxn。物理 I/O 寄存器和位定义列于 P75”I/O 端口寄存器的说明 ” 。

每个端口都有三个 I/O 存储器地址 : 数据寄存器 – PORTx、数据方向寄存器 – DDRx 和端口输入引脚 – PINx。数据寄存器和数据方向寄存器为读 / 写寄存器，而端口输入引脚为只读寄存器。但是需要特别注意的是，对 PINx 寄存器的某一位写入逻辑 "1“ 将造成数据寄存器相应位的数据发生 "0“ 与 “1“ 的交替变化。当寄存器 MCUCR 的上拉禁止位 PUD 置位时所有端口的全部引脚的上拉电阻都被禁止。

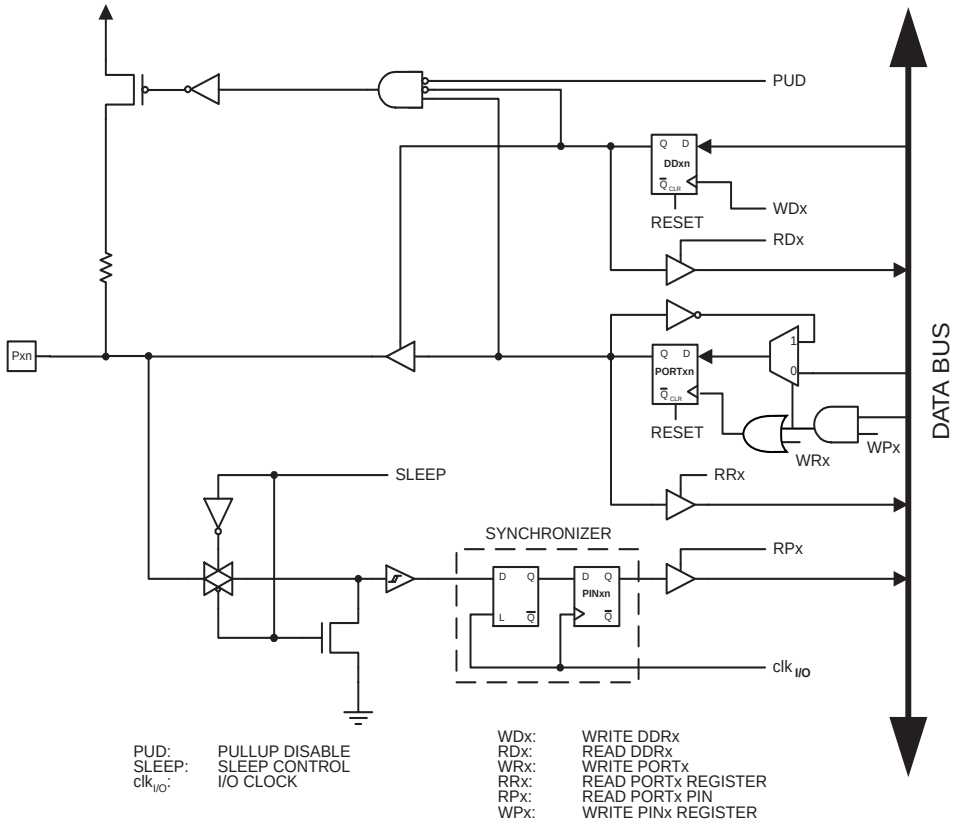
作为通用数字 I/O 时的端口请参见 P59” 作为通用数字 I/O 的端口 ” 。多数端口引脚是与第二功能复用的，如 P64” 端口的第二功能 ” 。请参见各个模块的具体说明以了解引脚的第二功能。

使能某些引脚的第二功能不会影响其他属于同一端口的引脚用于通用数字 I/O 的目的。

作为通用数字 I/O 的端口

端口为具有可选上拉电阻的双向 I/O 端口。Figure24 为 Pxn 的 I/O 端口引脚的说明。

Figure 24. 通用数字 I/O(1)



Note: 1. WRx, WPx, WDx, RRx, RPx 和 RDx 对于同一端口的所有引脚都是一样的。 $clk_{I/O}$, SLEEP 和 PUD 则对所有的端口都是一样的。

配置引脚

每个端口引脚都具有三个寄存器位 ：DDxn、PORTxn 和 PINxn，如 P75”I/O 端口寄存器的说明 ” 所示。 DDxn 位于 DDRx 寄存器， PORTxn 位于 PORTx 寄存器， PINxn 位于 PINx 寄存器。

DDxn 用来选择引脚的方向。 DDxn 为 "1“ 时， Pxn 配置为输出；否则为输入。

引脚配置为输入时，若 PORTxn 为 "1“，上拉电阻使能。如果需要关闭这个上拉电阻，可以将 PORTxn 清零，或者将这个引脚配置为输出。复位时各引脚为高阻态，即使此时并没有时钟在运行。

当引脚配置为输出时，若 PORTxn 为 "1“，引脚输出高电平 ("1“)，否则输出低电平 (“0“)。

使引脚电平交替变化

不论 DDRxn 是什么内容，向 PINxn 写逻辑 "1" 就会使 PORTxn 的值在 “0“ 和 "1“ 之间来回变化。注意 SBI 指令能够用来改变端口的单个位。

输入与输出之间的切换

在 (高阻态) 三态 ({DDxn, PORTxn} = 0b00) 和输出高电平 ({DDxn, PORTxn} = 0b11) 两种状态之间进行切换时，上拉电阻使能 ({DDxn, PORTxn} = 0b01) 或输出低电平 ({DDxn, PORTxn} = 0b10) 这两种模式必然会有一个发生。通常，上拉电阻使能是完全可以接受的，因为高阻环境并不在乎是强高电平输出还是上拉输出。如果实际应用环境不允许这样，则可以通过置位 MCUCR 寄存器的 PUD 来禁止所有端口的上拉电阻。

在上拉输入和输出低电平之间切换也有同样的问题。用户必须选择高阻态 ({DDxn, PORTxn} = 0b00) 或输出高电平 ({DDxn, PORTxn} = 0b10) 作为中间步骤。

Table32 总结了引脚的控制信号。

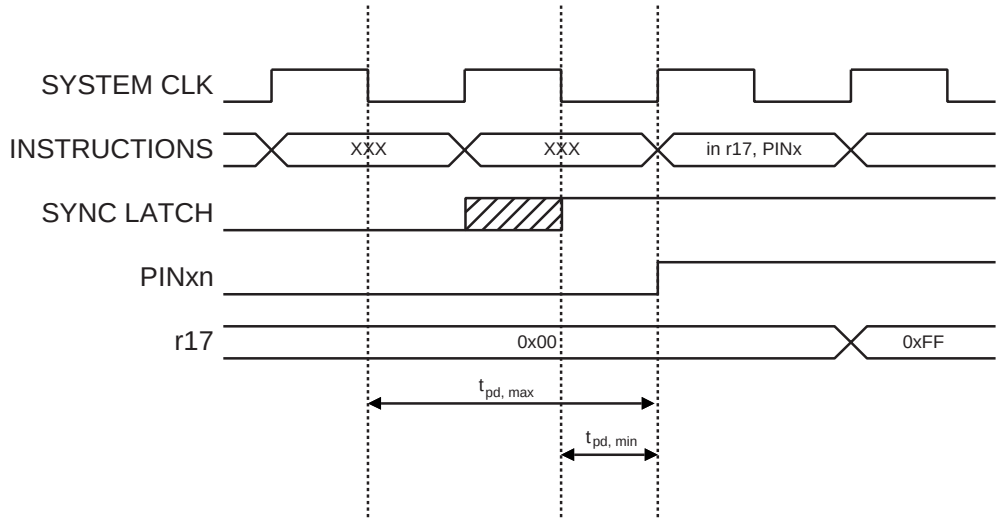
Table 32. 端口引脚配置

DDxn	PORTxn	PUD(位于 MCUCR)	I/O	上拉电阻	说明
0	0	X	输入	No	高阻态 (Hi-Z)
0	1	0	输入	Yes	被外部电路拉低时将输出电流
0	1	1	输入	No	高阻态 (Hi-Z)
1	0	X	输出	No	输出低电平 (吸收电流)
1	1	X	输出	No	输出高电平 (输出电流)

读取引脚上的数据

不论 DDxn 是如何配置的，都可以通过读取 PINxn 寄存器来获得引脚电平的信息。如 Figure24 所示，PINxn 寄存器的各个位与其前面的锁存器组成了一个同步器。这样就可以避免引脚电平在内部时钟边沿发生改变而造成的读取信号不稳定。其缺点是引入了延迟。Figure25 为读取引脚电平时同步器的时序图。最大和最小传输延迟分别为 $t_{pd,max}$ 和 $t_{pd,min}$。

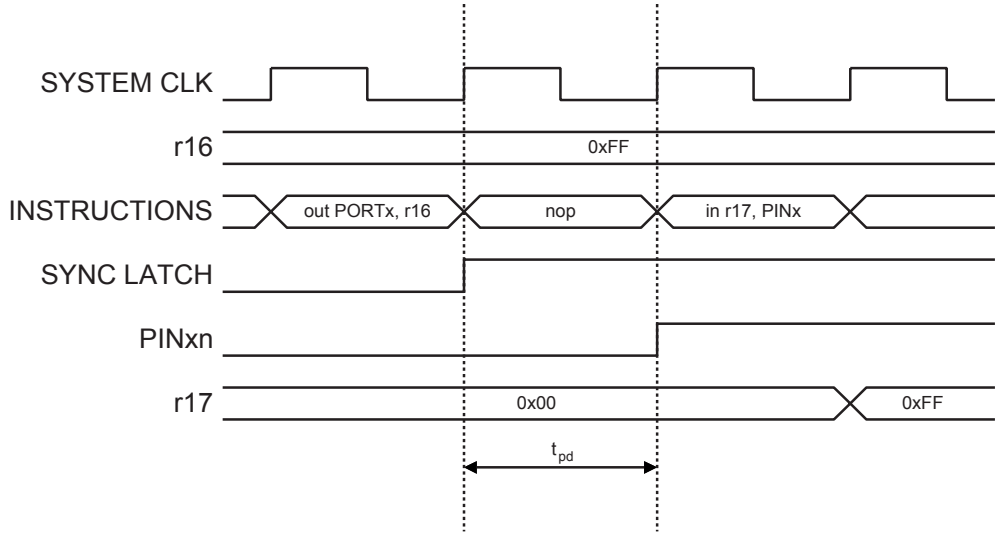
Figure 25. 读取引脚数据时的同步情况



考虑第一个系统时钟下降沿之后起始的时钟周期。当时钟信号为低时锁存器是关闭的；而时钟信号为高时信号可以自由通过，如图中 SYNC LATCH 信号的阴影区所示。时钟为低时信号即被锁存，然后在紧接着的系统时钟上升沿锁存到 PINxn 寄存器。如 $t_{pd,max}$ 和 $t_{pd,min}$ 所示，根据信号加入时间的不同，引脚上的信号转换延迟界于½ 到1½ 个系统时钟。

如 Figure26 所示，读取软件赋予的引脚电平时需要在赋值指令 *out* 和读取指令 *in* 之间至少有一个时钟周期的间隔，如 *nop* 指令。*out* 指令在时钟的上升沿置位 SYNC LATCH 信号。此时同步器的延迟时间 t_{pd} 为一个系统时钟。

Figure 26. 读取软件赋予的引脚电平的同步情况



下面的例程演示了如何置位端口 B 的引脚 0 和 1，引脚 2 和 3 输出低电平，以及将引脚 4 到 7 设置为输入，并且为引脚 6 和 7 设置上拉电阻。然后程序将各个引脚的数据读回来。如前面讨论的那样，我们在输出和输入语句之间插入了一个 *nop* 指令。

汇编代码例程 (1)

```
...
;定义上拉电阻和设置高电平输出
; 为端口引脚定义方向
ldi   r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi   r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out   PORTB,r16
out   DDRB,r17
; 为了同步插入 nop 指令
nop
; 读取端口引脚
in    r16,PINB
...
```

C 代码例程

```
unsigned char i;
  ...
  /* 定义上拉电阻和设置高电平输出 */
  /* 为端口引脚定义方向 */
  PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
  DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
  /* 为了同步插入 nop 指令 */
  _NOP();
  /* 读取端口引脚 */
  i = PINB;
  ...
```

Note: 1. 在汇编程序里使用了两个暂存器。其目的是为了使引脚0、1、6与7上拉过程的时间最短。当方向位设置正确后，定义 2、 3 位为低，且重新将 0 与 1 位定义为强驱动器。

数字输入使能和休眠模式

如 Figure24 所示，数字输入信号 (施密特触发器的输入) 可以钳位到地。图中的 SLEEP 信号由 MCU 的休眠控制器在各种掉电模式、省电模式以及 Standby 模式下设置，以防止在输入悬空或模拟输入电平接近 $V_{CC}/2$ 时消耗太多的电流。

引脚作为外部中断输入时 SLEEP 信号无效。但若外部中断没有使能， SLEEP 信号仍然有效。在引脚的第二功能使能时 SLEEP 也让位于第二功能，如 P64" 端口的第二功能 " 里描述的那样。

如果逻辑高电平 ("1") 出现在一个被设置为 " 上升沿中断、下降沿中断或任何逻辑电平变化都引起中断 " 的外部异步中断引脚上，即使该外部中断未被使能，但从上述休眠模式唤醒时，相应的外部中断标志位仍会被置 "1"。这是因为引脚电平在休眠模式下被钳位到 "0" 电平。唤醒过程造成了引脚电平从 "0" 到 "1" 的变化。

未连接引脚的处理

如果有引脚未被使用，建议给这些引脚赋予一个确定电平。虽然如上文所述，在深层休眠模式下大多数数字输入被禁用，但还是需要避免因引脚没有确定的电平而造成悬空引脚在其它数字输入使能模式 (复位、工作模式、空闲模式) 消耗电流。

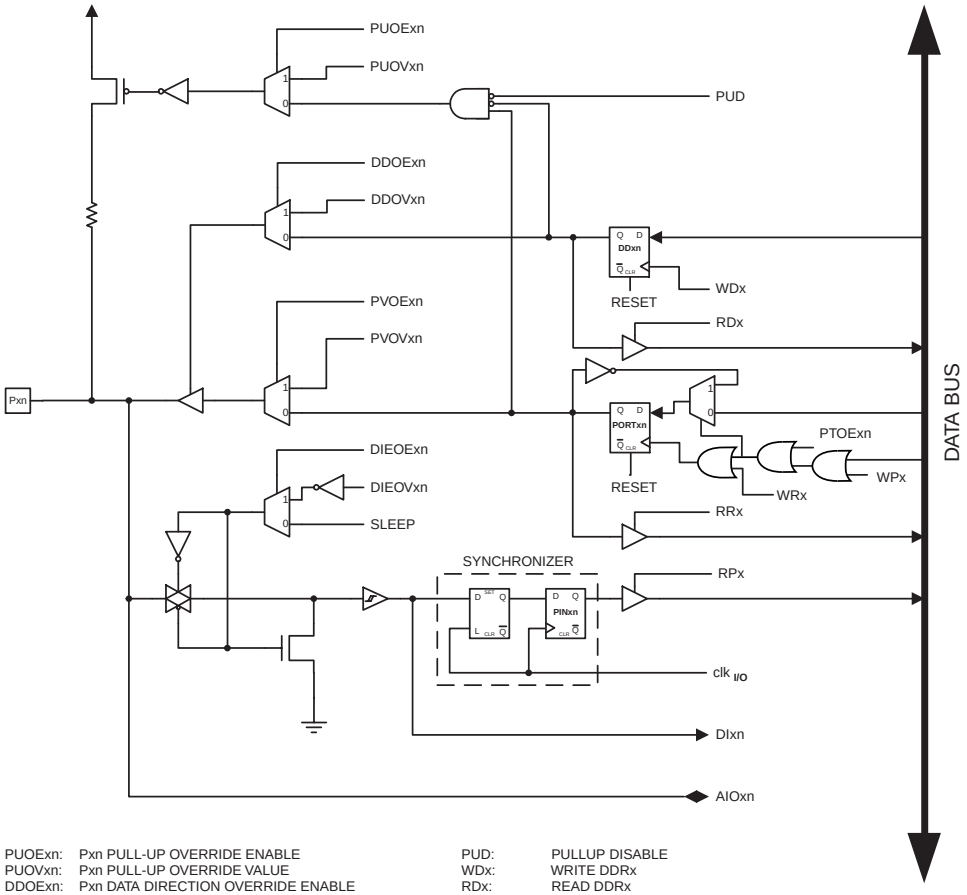
最简单的保证未用引脚具有确定电平的方法是使能内部上拉电阻。但要注意的是复位时上拉电阻将被禁用。如果复位时的功耗也有严格要求则建议使用外部上拉或下拉电阻。不

推荐直接将未用引脚与 V_{CC} 或 GND 连接，因为这样可能会在引脚偶然作为输出时出现冲击电流。

端口的第二功能

除了一般的数字 I/O 之外，大多数端口引脚都具有第二功能。Figure27 说明了由 Figure24 简化出来的端口引脚控制信号是如何被第二功能所重载的。这些被重载的信号不会出现在所有的端口引脚，但本图可以看作是适用于AVR系列处理器所有端口引脚的一般说明。

Figure 27. 端口的第二功能 (1)



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE	PUD:	PULLUP DISABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE	WDx:	WRITE DDRx
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE	RDx:	READ DDRx
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE	RRx:	READ PORTx REGISTER
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE	WRx:	WRITE PORTx
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE	RPx:	READ PORTx PIN
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE	WPx:	WRITE PINx
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE	$clk_{I/O}$:	I/O CLOCK
SLEEP:	SLEEP CONTROL	DIxn:	DIGITAL INPUT PIN n ON PORTx
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE	AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

Note: 1. WPx, WDx, RLx, RPx和RDx对于同一个端口的所有引脚都是一样的。$clk_{I/O}$, SLEEP和PUD 则对所有的端口都是一样的。其他信号只对某一个引脚有效。

Table33 为重载信号的简介。表中没有给出 Figure27 的引脚和端口索引。这些重载信号是由第二功能模块内部产生的。

Table 33. 第二功能重载信号的一般说明

信号名称	全 称	说 明
PUOE	上拉电阻 重载使能	若此信号置位，上拉电阻使能将受控于 PUOV ；若此信号清零，则 {DDxn, PORTxn, PUD} = 0b010 时上拉电阻使能。
PUOV	上拉电阻 重载使能	若 PUOE 置位，则 PUOV 置位 / 清零时上拉电阻使能 / 禁止，而不管 DDxn、PORTxn 和 PUD 寄存器各个位的设置如何。
DDOE	数据方向 重载使能	如果此信号置位，则输出驱动使能由 DDOV 控制；若此信号清零，输出驱动使能由 DDxn 寄存器控制。
DDOV	数据方向 重载使能	若 DDOE 置位，则 DDOV 置位 / 清零时输出驱动使能 / 禁止，而不管 DDxn 寄存器的设置如何。
PVOE	端口数据 重载使能	如果这个信号置位，且输出驱动使能，端口数据由 PVOV 控制；若 PVOE 清零，且输出驱动使能，端口数据由寄存器 PORTxn 控制。
PVOV	端口数据 重载使能	若 PVOE 置位，端口值设置为 PVOV，而不管寄存器 PORTxn 如何设置。
PTOE	端口信号切换 重载使能	若 PTOE 置位，端口寄存器位取反。
DIEOE	数字输入 使能覆盖使能	如果这个信号置位，数字输入使能由 DIEOV 控制；若 DIEOE 清零，数字输入使能由 MCU 的状态确定 (正常模式，睡眠模式)。
DIEOV	数字输入 使能覆盖使能	若 DIEOE 置位， DIEOV 置位 / 清零时数字输入使能 / 禁止，而不管 MCU 的状态如何 (正常模式，睡眠模式)。
DI	数字输入	此信号为第二功能的数字输入。在图中，这个信号与施密特触发相连，并且在同步器之前。除非数字输入用作时钟源，否则第二功能模块将使用自己的同步器。
AIO	模拟信号 输入 / 输出	模拟输入 / 输出。信号直接与引脚接点相连，而且可以用作双向端口。

下面的几小节将简单地说明每个端口的第二功能以及相关的信号。具体请参考有关第二功能的说明。

MCU 控制寄存器—MCUCR

位	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
读 / 写	R/W	R	R	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 4 – PUD: 禁用上拉电阻**

PUD 置位时，即使寄存器 DDxn 和 PORTxn 配置为使能上拉电阻 ({DDxn, PORTxn} = 0b01)，I/O 端口的上拉电阻也被禁止。请参见 P60” 配置引脚 ” 。

端口 B 的第二功能

端口 B 的第二功能列于 Table34.

Table 34. 端口 B 的第二功能

端口引脚	第二功能
PB7	XTAL2 (芯片时钟振荡器引脚 2) TOSC2 (定时器振荡器引脚 2) PCINT7 (引脚电平变化中断 7)
PB6	XTAL1 (芯片时钟振荡器引脚 1 或外部时钟输入) TOSC1 (定时电平器振荡器引脚 1) PCINT6 (引脚变化中断 6)
PB5	SCK (SPI 总线主时钟输入) PCINT5 (引脚变化中断 5)
PB4	MISO (SPI 总线主机输入 / 从机输出) PCINT4 (引脚电平变化中断 4)
PB3	MOSI (SPI 总线主输出 / 从输入) OC2A (定时器 / 计数器 2 输出比较匹配 A 输出) PCINT3 (引脚电平变化中断 3)
PB2	$\overline{SS}$ (SPI 总线主从选择) OC1B (定时器 / 计数器 1 输出比较匹配 B 输出) PCINT2 (引脚电平变化中断 2)
PB1	OC1A (定时器 / 计数器 1 输出比较匹配 A 输出) PCINT1 (引脚电平变化中断 1)
PB0	ICP1 (定时器 / 计数器 1 输入捕捉输入) CLKO (系统时钟分频输出) PCINT0 (引脚电平变化中断 0)

引脚配置如下：

- **XTAL2/TOSC2/PCINT7 – 端口 B, 位 7**

XTAL2 : 芯片时钟振荡器引脚 2。作为晶振或低频晶振的时钟引脚。作为时钟引脚时不能作为 I/O 引脚。

TOSC2 : 定时器振荡器引脚 2。只有选择了内部标定的 RC 振荡器作为系统时钟源，而且设置了寄存器 ASSR 的 AS2 位以允许使用异步时钟定时器时才可使用。当 ASSR 寄存器的 AS2 位置位且 EXCLK 位清零，从而使能了定时器 / 计数器 2 的异步时钟功能时，PB7 与端口引脚脱离，作为振荡放大器的反向输出端。在该模式下，时钟晶体连接到该引脚，且不能作为 I/O 引脚。

PCINT7：引脚电平变化中断源 7。PB7 可以作为外部中断源。

如果 PB7 作为时钟引脚使用， DDB7、PORTB7 与 PINB7 的读返回值为 0。

- **XTAL1/TOSC1/PCINT6 – 端口 B, 位 6**

XTAL1 ：芯片时钟振荡器引脚 1。可用于除内部标定 RC 振荡器外的所有时钟源。作为时钟引脚时不能作为 I/O 引脚。

TOSC1 : 定时器振荡器引脚 1。只有选择了内部标定的 RC 振荡器作为系统时钟源，而且设置了寄存器 ASSR 的 AS2 位以允许使用异步时钟定时器时才可使用。当 ASSR 寄存器的 AS2 位置位，从而使能了定时器 / 计数器 2 的异步时钟功能时，PB6 与端口引脚脱离，作为振荡放大器的反向输入端。在该模式下，时钟晶体连接到该引脚，且不能作为 I/O 引脚。

PCINT6 ：引脚电平变化中断源 6。 PB6 可以作为外部中断源。

如果 PB6 作为时钟引脚使用， DDB6、PORTB6 与 PINB6 均读为 "0”。

- **SCK/PCINT5 – 端口 B, 位 5**

SCK : 用于使用 SPI 串行总线接口。当 SPI 使能且为从机时，无论 DDB5 为何种设置，该引脚被强置为输入。当使能 SPI 且为主机时，该引脚的数据方向由 DDB5 来控制。即使该位被 SPI 强制为输入，但内部上拉电阻仍然由 PORTB5 来控制。

PCINT5 ：引脚电平变化中断源 5。 PB5 可以作为外部中断源。

- **MISO/PCINT4 – 端口 B, 位 4**

MISO :SPI 总线接口的主机数据输入 / 从机数据输出端。在 SPI 使能，且工作于 SPI 主机模式时，无论 DDB4 为何值， PB4 被设置为输入；为 SPI 从机模式时，该引脚的数据方向由 DDB4 控制。当该引脚被 SPI 强制为输入时，内部上拉电阻仍然由 PORTB4 来控制。

PCINT4 ：引脚电平变化中断源 4。 PB4 可以作为外部中断源。

- **MOSI/OC2/PCINT3 – 端口 B, 位 3**

MOSI :SPI 总线接口的主机数据输出 / 从机数据输入端。在 SPI 使能，且工作于从机模式时，无论 DDB3 为何值， PB3 被设置为输入；为 SPI 主机模式时，该引脚的数据方向由 DDB3 控制。当该引脚被 SPI 强制为输入时，内部上拉电阻仍然由 PORTB3 来控制。

OC2，输出比较匹配输出。PB3 引脚可作为定时器 / 计数器 2 比较匹配的外部输出口。此时， PB3 引脚必须设置为输出 (DDB3=1) 。在 PWM 应用中， OC2 引脚还作为 PWM 定时器模块的输出引脚。

PCINT3 ：引脚电平变化中断源 3。 PB3 可以作为外部中断源。

- **$\overline{\text{SS}}$/OC1B/PCINT2 – 端口 B, 位 2**

$\overline{\text{SS}}$:从机选择输入。在 SPI 使能，且工作于从机模式时，无论 DDB2 为何值，PB2 脚被设置为输入。当 PB2 被外部拉低时，则 SPI 功能被激活。当使能 SPI，且为主机模式时该引脚的数据方向由 DDB2 控制。当该引脚被 SPI 强制为输入时，内部上拉电阻仍然由 PORTB2 来控制。

OC1B，输出比较匹配输出。PB2 引脚可作为定时器 / 计数器 1 比较匹配 B 的外部输出口。此时，PB2 引脚必须设置为输出 (DDB2=1) 。在 PWM 应用中，OC1B 引脚还作为 PWM 定时器模块的输出引脚。

PCINT2：引脚电平变化中断源 2。 PB2 可以作为外部中断源。

- **OC1A/PCINT1 – 端口 B, 位 1**

OC1A，输出比较匹配输出。PB1 引脚可作为定时器 / 计数器 1 比较匹配 A 的外部外部输出口。此时， PB1 引脚必须设置为输出 (DDB1=1) 。在 PWM 应用中， OC1A 引脚还作为 PWM 定时器模块的输出引脚。

PCINT1：引脚电平变化中断源 1。 PB1 可以作为外部中断源。

- **ICP1/CLKO/PCINT0 – 端口 B, 位 0**

ICP1，输入捕获引脚：PB0 引脚可以作为定时器 / 计数器 1 的输入捕获功能的引脚。

CLKO ,分频之后的系统时钟:分频之后的系统时钟可以通过 PB0 引脚输出。如果 CKOUT 熔丝位编程，无论 PORTB0 与 DDB0 为何值，分频之后的系统时钟都将从此引脚输出。复位时时钟信号照样从此引脚输出。

PCINT0，引脚电平变化中断源 0 ：PB0 可以作为外部中断源。

Table35 和Table36 将端口 B的第二功能与P63Figure 27 的重载信号关联在了一起。SPI MSTR INPUT 和SPI SLAVE OUTPUT 构成了MISO信号 ,而MOSI可以分解为SPI MSTR OUTPUT 和 SPI SLAVE INPUT。

Table 35. PB7..PB4 的第二功能重载信号

信号名称	PB7/XTAL2/ TOSC2/PCINT7(1)	PB6/XTAL1/ TOSC1/PCINT6(1)	PB5/SCK/ PCINT5	PB4/MISO/ PCINT4
PUOE	$\overline{\text{INTRC}}$ • $\overline{\text{EXTCK}}$+ AS2	$\overline{\text{INTRC}}$ + AS2	SPE • $\overline{\text{MSTR}}$	SPE • MSTR
PUOV	0	0	PORTB5 • $\overline{\text{PUD}}$	PORTB4 • $\overline{\text{PUD}}$
DDOE	$\overline{\text{INTRC}}$ • $\overline{\text{EXTCK}}$+ AS2	$\overline{\text{INTRC}}$ + AS2	SPE • $\overline{\text{MSTR}}$	SPE • MSTR
DDOV	0	0	0	0
PVOE	0	0	SPE • MSTR	SPE • $\overline{\text{MSTR}}$
PVOV	0	0	SCK OUTPUT	SPI SLAVE OUTPUT
DIEOE	$\overline{\text{INTRC}}$ • $\overline{\text{EXTCK}}$ + AS2 + PCINT7 • PCIE0	$\overline{\text{INTRC}}$ + AS2 + PCINT6 • PCIE0	PCINT5 • PCIE0	PCINT4 • PCIE0
DIEOV	(INTRC + EXTCK) • $\overline{\text{AS2}}$	INTRC • $\overline{\text{AS2}}$	1	1
DI	PCINT7INPUT	PCINT6INPUT	PCINT5 INPUT SCK INPUT	PCINT4 INPUT SPI MSTRINPUT
AIO	Oscillator Output	Oscillator/Clock Input	–	–

Notes: 1. INTRC表示选择一个内部RC振荡器(通过CKSEL 熔丝位)，EXTCK表示选择外部时钟(通过 CKSEL 熔丝位)

Table 36. PB3..PB0 的第二功能重载信号

信号名称	PB3/MOSI/ OC2/PCINT3	PB2/$\overline{SS}$/ OC1B/PCINT2	PB1/OC1A/ PCINT1	PB0/ICP1/ PCINT0
PUOE	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	0	0
PUOV	PORTB3 • $\overline{PUD}$	PORTB2 • $\overline{PUD}$	0	0
DDOE	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	0	0
DDOV	0	0	0	0
PVOE	SPE • MSTR + OC2A ENABLE	OC1B ENABLE	OC1AENABLE	0
PVOV	SPI MSTR OUTPUT + OC2A	OC1B	OC1A	0
DIEOE	PCINT3 • PCIE0	PCINT2 • PCIE0	PCINT1 • PCIE0	PCINT0 • PCIE0
DIEOV	1	1	1	1
DI	PCINT3 INPUT SPI SLAVE INPUT	PCINT2 INPUT SPI $\overline{SS}$	PCINT1 INPUT	PCINT0 INPUT ICP1 INPUT
AIO	–	–	–	–

端口 C 的第二功能

端口 C 的第二功能见 Table37。

Table 37. 端口 C 的第二功能

端口引脚	第二功能
PC6	$\overline{RESET}$ (复位引脚) PCINT14 (引脚电平变化中断 14)
PC5	ADC5 (ADC 输入通道 5) SCL (两线串行总线接口时钟线) PCINT13 (引脚电平变化中断 13)
PC4	ADC4 (ADC 输入通道 4) SDA (两线串行总线接口数据输入 / 输出线) PCINT12 (引脚电平变化中断 12)
PC3	ADC3 (ADC 输入通道 3) PCINT11 (引脚电平变化中断 11)
PC2	ADC2 (ADC 输入通道 2) PCINT10 (引脚电平变化中断 10)
PC1	ADC1 (ADC 输入通道 1) PCINT9 (引脚电平变化中断 9)
PC0	ADC0 (ADC 输入通道 0) PCINT8 (引脚电平变化中断 8)

引脚配置如下：

- **$\overline{\text{RESET}}$/PCINT14 – 端口 C, 位 6**

$\overline{\text{RESET}}$,复位引脚:熔丝位 RSTDISBL 被编程时 ,该引脚作为普通的 I/O 引脚使用 ,此时 ,芯片内部的上电复位与 BOD 复位电路将作为系统的复位源。当 RSTDISBL 熔丝位被清零时，内部复位电路将连接到该引脚，此时引脚不作为 I/O 使用。

如果 PC6 工作为复位引脚， DDC6、PORTC6 与 PINC6 的读返回值为 0。

PCINT14 ：引脚电平变化中断源 14。 PC6 可作为外部中断源。

- **SCL/ADC5/PCINT13 – 端口 C, 位 5**

SCL ,两线串行总线的时钟线:当 TWCR 寄存器中的 TWEN 位被设为 "1" ,使能 TWI 接口时， PC5 引脚将与 I/O 端口脱离，成为 TWI 总线接口的串行时钟线。在该模式下，有一个尖峰滤波器连接到该引脚，能够抑制输入信号中小于 50 ns 的毛刺，同时引脚由具有上升率限制的开漏驱动器驱动。

PC5 也可作为 ADC 输入通道 5。注意， ADC 输入通道 5 由数字电源供电。

PCINT13：引脚电平变化中断源 13。 PC5 可作为外部中断源。

- **SDA/ADC4/PCINT12 – 端口 C, 位 4**

SDA ,两线串行总线数据线:当 TWCR 寄存器中的 TWEN 位被设为 1 ,使能 TWI 接口时，PC5引脚将与I/O端口脱离 ,成为TWI总线接口的串行数据线。在该模式下 ,有一个尖峰滤波器连接到该引脚 ,能够抑制输入信号中小于 50 ns 的毛刺 ,同时引脚将由具有上升率限制的开漏驱动器驱动。

PC4 也可作为 ADC 输入通道 4。注意， ADC 输入通道 4 由数字电源供电。

PCINT12：引脚电平变化中断源 12。 PC4 可作为外部中断源。

- **ADC3/PCINT11 – 端口 C, 位 3**

PC3 也可作为 ADC 输入通道 3。注意， ADC 输入通道 3 由模拟电源供电。

PCINT11：引脚电平变化中断源 11。 PC3 可作为外部中断源。

- **ADC2/PCINT10 – 端口 C, 位 2**

PC2 也可作为 ADC 输入通道 2。注意， ADC 输入通道 2 由模拟电源供电。

PCINT10：引脚电平变化中断源 10。 PC2 可作为外部中断源。

- **ADC1/PCINT9 – 端口 C, 位 1**

PC1 也可作为 ADC 输入通道 1。注意， ADC 输入通道 1 由模拟电源供电。

PCINT9：引脚电平变化中断源 9。 PC1 可作为外部中断源。

- **ADC0/PCINT8 – 端口 C, 位 0**

PC0 也可作为 ADC 输入通道 0。注意， ADC 输入通道 0 由模拟电源供电。

PCINT8：引脚电平变化中断源 8。 PC0 可作为外部中断源。

Table38 和 Table39 将端口 C 的第二功能与 P63Figure 27 的重载信号关联在了一起。

Table 38. PC6..PC4[1] 的第二功能重载信号

信号名称	PC6/$\overline{\text{RESET}}$/PCINT14	PC5/SCL/ADC5/PCINT13	PC4/SDA/ADC4/PCINT12
PUOE	$\overline{\text{RSTDISBL}}$	TWEN	TWEN
PUOV	1	PORTC5 • $\overline{\text{PUD}}$	PORTC4 • $\overline{\text{PUD}}$
DDOE	$\overline{\text{RSTDISBL}}$	TWEN	TWEN
DDOV	0	SCL_OUT	SDA_OUT
PVOE	0	TWEN	TWEN
PVOV	0	0	0
DIEOE	$\overline{\text{RSTDISBL}}$ + PCINT14 • PCIE1	PCINT13 • PCIE1 + ADC5D	PCINT12 • PCIE1 + ADC4D
DIEOV	RSTDISBL	PCINT13 • PCIE1	PCINT12 • PCIE1
DI	PCINT14 INPUT	PCINT13 INPUT	PCINT12 INPUT
AIO	RESET INPUT	ADC5 INPUT / SCL INPUT	ADC4 INPUT / SDA INPUT

Note: 1. 使能时 ,两线串行接口为引脚 PC4 和 PC5 的输出设置了上升率控制。这在表中没有显示。此外，尖峰滤波器位于 AIO 与 TWI 模块的数字逻辑之间。

Table 39. PC3..PC0 的第二功能重载信号

信号名称	PC3/ADC3/ PCINT11	PC2/ADC2/ PCINT10	PC1/ADC1/ PCINT9	PC0/ADC0/ PCINT8
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT11 • PCIE1 + ADC3D	PCINT10 • PCIE1 + ADC2D	PCINT9 • PCIE1 + ADC1D	PCINT8 • PCIE1 + ADC0D
DIEOV	PCINT11 • PCIE1	PCINT10 • PCIE1	PCINT9 • PCIE1	PCINT8 • PCIE1
DI	PCINT11 INPUT	PCINT10 INPUT	PCINT9 INPUT	PCINT8 INPUT
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

端口 D 的第二功能

端口 D 的第二功能列于 Table40。

Table 40. 端口 D 的第二功能

端口引脚	第二功能
PD7	AIN1 (模拟比较器负输入) PCINT23 (引脚电平变化中断 23)
PD6	AIN0 (模拟比较器负输入) OC0A (定时器 / 计数器 0 输出比较匹配 A 输出) PCINT22 (引脚电平变化中断 22)
PD5	T1 (定时器 / 计数器 1 外部计数器输入) OC0B (定时器 / 计数器 0 输出比较匹配 B 输出) PCINT21 (引脚电平变化中断 21)
PD4	XCK (USART 外部时钟输入 / 输出) T0 (定时器 / 计数器 0 外部计数器输入) PCINT20 (引脚电平变化中断 20)
PD3	INT1 (外部中断 1 输入) OC2B (定时器 / 计数器 2 输出比较匹配 B 输出) PCINT19 (引脚电平变化中断 19)
PD2	INT0 (外部中断 0 输入) PCINT18 (引脚电平变化中断 18)
PD1	TXD (USART 输出引脚) PCINT17 (引脚电平变化中断 17)
PD0	RXD (USART 输入引脚) PCINT16 (引脚电平变化中断 16)

第二功能配置如下：

- **AIN1/OC2B/PCINT23 – 端口 D, 位 7**

AIN1 , 模拟比较器负输入 : 将 PD7 设置为输入 , 且关闭内部上拉电阻 , 以避免数字端口功能影响模拟比较器的性能。

PCINT23：引脚电平变化中断源 23。 PD7 可作为外部中断源。

- **AIN0/OC0A/PCINT22 – 端口 D, 位 6**

AIN0 , 模拟比较器正输入 : 将 PD6 设置为输入 , 且关闭内部上拉电阻 , 以避免数字端口功能影响模拟比较器的性能。

OC0A , 输出比较匹配输出 : PD6 引脚可作为定时器 / 计数器 0 比较匹配 A 的外部输出口。此时， PD6 引脚必须设置为输出 (DDD6=1) 。在 PWM 应用中， OC0A 引脚还作为 PWM 定时器模块的输出引脚。

PCINT22：引脚电平变化中断源 22。 PD6 可作为外部中断源。

- **T1/OC0B/PCINT21 – 端口 D, 位 5**

T1，定时器 / 计数器 1 计数器源。

OC0B , 输出比较匹配输出 : PD5 引脚可作为定时器 / 计数器 0 比较匹配 B 的外部输出口。此时， PD5 引脚必须设置为输出 (DDD5=1) 。在 PWM 应用中， OC0B 引脚还作为 PWM 定时器模块的输出引脚。

PCINT21：引脚电平变化中断源 21。 PD5 可作为外部中断源。

- **XCK/T0/PCINT20 – 端口 D, 位 4**

XCK，USART 外部时钟。

T0，定时器 / 计数器 0 计数器源。

PCINT20：引脚电平变化中断源 20。 PD4 可作为外部中断源。

- **INT1/OC2B/PCINT19 – 端口 D, 位 3**

INT1，外部中断源 1：PD3 引脚可作为外部中断源。

OC2B，输出比较匹配输出：PD3 引脚可作为定时器/计数器0比较匹配B的外部输出口。此时， PD3 引脚必须设置为输出 (DDD3=1) 。在 PWM 应用中， OC2B 引脚还作为 PWM 定时器模块的输出引脚。

PCINT19：引脚电平变化中断源 19。 PD3 可作为外部中断源。

- **INT0/PCINT18 – 端口 D, 位 2**

INT0，外部中断源 0：PD2 引脚可作为外部中断源。

PCINT18：引脚电平变化中断源 18。 PD2 可作为外部中断源。

- **TXD/PCINT17 – 端口 D, 位 1**

TXD，传输数据 (USART 的数据输出引脚)：当 USART 传输器使能，不论 DDD1 为何值，该引脚被配置为输出口。

PCINT17：引脚电平变化中断源 17。 PD1 可作为外部中断源。

- **RXD/PCINT16 – 端口 D, 位 0**

RXD，接收数据 (USART 的数据输入引脚)：当 USART 接收器使能，不论 DDD0 为何值，该引脚被配置为输入口，此时引脚的内部上拉功能仍然由 PORTD0 位控制。

PCINT16：引脚电平变化中断源 16。 PD0 可作为外部中断源。

Table41 和 Table42 将端口 D 的第二功能与 P63Figure 27 的重载信号关联在了一起。

Table 41. PD7..PD4 第二功能重载信号

信号名称	PD7/AIN1 /PCINT23	PD6/AIN0/ OC0A/PCINT22	PD5/T1/OC0B/ PCINT21	PD4/XCK/ T0/PCINT20
PUOE	0	0	0	0
PUO	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	OC0A ENABLE	OC0B ENABLE	UMSEL
PVOV	0	OC0A	OC0B	XCK OUTPUT
DIEOE	PCINT23 • PCIE2	PCINT22 • PCIE2	PCINT21 • PCIE2	PCINT20 • PCIE2
DIEOV	1	1	1	1
DI	PCINT23 INPUT	PCINT22 INPUT	PCINT21 INPUT T1 INPUT	PCINT20 INPUT XCK INPUT T0 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	–	–

Table 42. PD3..PD0 第二功能重载信号

信号名称	PD3/OC2B/INT1/ PCINT19	PD2/INT0/ PCINT18	PD1/TXD/ PCINT17	PD0/RXD/ PCINT16
PUOE	0	0	TXEN	RXEN
PUO	0	0	0	PORTD0 • $\overline{PUD}$
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	OC2B ENABLE	0	TXEN	0
PVOV	OC2B	0	TXD	0
DIEOE	INT1 ENABLE + PCINT19 • PCIE2	INT0 ENABLE + PCINT18 • PCIE1	PCINT17 • PCIE2	PCINT16 • PCIE2
DIEOV	1	1	1	1
DI	PCINT19 INPUT INT1 INPUT	PCINT18 INPUT INT0 输 INPUT 入	PCINT17 INPUT	PCINT16 INPUT RXD
AIO	–	–	–	–

I/O 端口寄存器的说明

端口 B 数据寄存器—PORTB

位	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 B 数据方向寄存器—DDRB

位	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 B 输入引脚地址—PINB

位	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 C 数据寄存器—PORTC

位	7	6	5	4	3	2	1	0	
	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
读 / 写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 C 数据方向寄存器—DDRC

位	7	6	5	4	3	2	1	0	
	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
读 / 写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 C 输入引脚地址—PINC

位	7	6	5	4	3	2	1	0	
	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读 / 写	R	R	R	R	R	R	R	R	
初始值	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 D 数据寄存器—PORTD

位	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 D 数据方向寄存器—DDRD

位	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 D 输入引脚地址—PIND

位	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

外部中断

外部中断通过引脚 INT0 与 INT1 或 PCINT23..0 触发。只要使能了中断，即使引脚 INT0 与 INT1 或 PCINT23..0 配置为输出，只要电平发生了合适的变化，中断也会触发。这个特点可以用来产生软件中断。只要使能， PCINT23..16 引脚上的电平变化将触发外部中断 PCI2， PCINT14..8 引脚上的电平变化将触发外部中断 PCI1， PCINT7..0 将触发外部中断 PCI0。PCMSK2、PCMSK1 与 PCMSK0 寄存器则用来检测是哪个引脚上的电平发生了变化。 PCINT23..0 外部中断的检测是异步的。也就是说，和其他中断方式一样，这些中断也可以用来将器件从休眠模式唤醒。

INT0 与 INT1 中断可以由下降沿、上升沿，或者是低电平触发。具体由外部中断控制寄存器 A – EICRA 的设置来确定。当 INT0 或 INT1 中断使能且设定为电平触发时，只要引脚电平被拉低，中断就会产生。若要求 INT0 或 INT1 在信号下降沿或上升沿触发中断，则 I/O时钟必须工作(请参见P23”时钟系统及其分布” 了解更多信息)。INT0与INT1 的低电平中断检测是异步的。也就是说它可以用来将器件从休眠模式唤醒。在休眠过程 (除了空闲模式) 中 I/O 时钟是停止的。

通过电平中断将 MCU 从掉电模式唤醒时，要保证低电平保持一定的时间以使 MCU 完成唤醒过程并触发中断。如果触发电平在启动时间结束前就消失， MCU 将被唤醒，但中断不会被触发。启动时间由熔丝位 SUT 与 CKSEL 决定。详见 P23” 系统时钟及其选项 ” 。

外部中断控制寄存器 A—EICRA

外部中断控制寄存器 A 包括决定中断触发方式的控制位。

位	7	6	5	4	3	2	1	0	
	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..4 – Res: 保留位**

ATmega48/88/168 没有使用这些位，其读返回值为 "0”。

- **位 3, 2 – ISC11, ISC10: 中断触发方式控制 1 位 1 与位 0**

外部中断 1 由引脚 INT1 激发，如果 SREG 寄存器的 I 标志位和相应的中断屏蔽位置位的话。触发方式如 Table43 所示。在检测边沿前 MCU 首先采样 INT1 引脚上的电平。如果选择了边沿触发方式或电平变化触发方式，那么持续时间大于一个时钟周期的脉冲将触发中断，过短的脉冲则不能保证触发中断。如果选择低电平触发方式，那么低电平必须保持到当前指令执行完成。

Table 43. 中断 1 触发方式控制

ISC11	ISC10	说明
0	0	INT1 为低电平时产生中断请求
0	1	INT1 引脚上任意的逻辑电平变化都将引发中断
1	0	INT1 的下降沿产生异步中断请求
1	1	INT1 的上升沿产生异步中断请求

- **位 1, 0 – ISC01, ISC00: 中断触发方式控制 0 之位 1 与位 0**

外部中断 0 由引脚 INT0 激发，如果 SREG 寄存器的 I 标志位和相应的中断屏蔽位置位的话。触发方式如 Table44 所示。在检测边沿前 MCU 首先采样 INT0 引脚上的电平。如果选择了边沿触发方式或电平变化触发方式，那么持续时间大于一个时钟周期的脉冲将触发中断，过短的脉冲则不能保证触发中断。如果选择低电平触发方式，那么低电平必须保持到当前指令执行完成。

Table 44. 中断 0 触发方式控制

ISC01	ISC00	说明
0	0	INT0 为低电平时产生中断请求
0	1	INT0 引脚上任意的逻辑电平变化都将引发中断
1	0	INT0 的下降沿产生异步中断请求
1	1	INT0 的上升沿产生异步中断请求

外部中断屏蔽寄存器—EIMSK

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	INT1	INT0	EIMSK
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..2 – Res: 保留位**

ATmega48/88/168 没有使用这些位，其读返回值为 "0”。

- **位 1 – INT1: 外部中断请求 1 使能**

当 INT1 为 ”1”，而且状态寄存器 SREG 的 I 标志置位，相应的外部引脚中断就使能了。外部中断控制寄存器 – EICRA 的中断触发方式控制位 (ISC11 与 ISC10) 决定中断是由 INT1 上升沿、下降沿，还是电平触发的。使能之后，即使引脚 INT1 被配置为输出，只要引脚电平发生了相应的变化，中断就会产生。相应的中断向量为 INT1 中断向量。

- **位 0 – INT0: 外部中断请求 0 使能**

当 INT0 为 ”1”，而且状态寄存器 SREG 的 I 标志置位，相应的外部引脚中断就使能了。外部中断控制寄存器 – EICRA 的中断触发方式控制位 (ISC11 与 ISC10) 决定中断是由 INT0 上升沿、下降沿，还是电平触发的。使能之后，即使引脚 INT0 被配置为输出，只要引脚电平发生了相应的变化，中断就会产生。相应的中断向量为 INT0 中断向量。

外部中断标志寄存器—EIFR

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	INTF1	INTF0	EIFR
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..2 – Res: 保留位**

ATmega48/88/168 没有使用这些位，其读返回值为 "0”。

- **位 1 – INTF1: 外部中断标志 1**

INT1 引脚电平发生跳变时触发中断请求，并置位相应的中断标志 INTF1。如果 SREG 的位 I 以及 EIMSK 寄存器相应的中断使能位 INT1 为 ”1” ，MCU 即跳转到相应的中断向量。进入中断服务程序之后该标志自动清零。此外，标志位也可以通过写入 ”1” 来清零。若 INT1 配置为电平触发，则 INTF1 一直为零。

- **位 0 – INTF0: 外部中断标志 0**

INT0 引脚电平发生跳变时触发中断请求，并置位相应的中断标志 INTF0。如果 SREG 的位 I 以及 EIMSK 寄存器相应的中断使能位 INT0 为 ”1” ，MCU 即跳转到相应的中断向量。进入中断服务程序之后该标志自动清零。此外，标志位也可以通过写入 ”1” 来清零。若 INT0 配置为电平触发，则 INTF0 一直为零。

引脚电平变化中断控制寄存器— PCICR

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
读 / 写	R	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..3 - Res: 保留位**

ATmega48/88/168 没有使用这些位，其读返回值为 "0”。

- **位 2 - PCIE2: 引脚电平变化中断使能 2**

当 PCIE2 位与 SREG 的位 I 置 "1”。使能的 PCINT23..16 引脚上的任何电平变化都会引起中断。相应的引脚电平变化中断请求由 PCI2 中断向量执行。PCINT23..16 引脚可以通过 PCMSK2 寄存器单独使能。

- **位 1 - PCIE1: 引脚电平变化中断使能 1**

当 PCIE1 位与 SREG 的位 I 置 "1”。使能的 PCINT14..8 引脚上的任何电平变化都会引起中断。相应的引脚电平变化中断请求由 PCI1 中断向量执行。PCINT14..8 引脚可以通过 PCMSK1 寄存器单独使能。

- **位 0 - PCIE0: 引脚电平变化中断使能 0**

当 PCIE0 位与 SREG 的位 I 置 "1”。使能的 PCINT7..0 引脚上的任何电平变化都会引起中断。相应的引脚电平变化中断请求由 PCI0 中断向量执行。PCINT7..0 引脚可以通过 PCMSK0 寄存器单独使能。

引脚电平变化中断标志寄存器— PCIFR

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
读 / 写	R	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..3 - Res: 保留位**

ATmega48/88/168 没有使用这些位，其读返回值为 "0”。

- **位 2 - PCIF2: 引脚电平变化中断标志 2**

当引脚 PCINT23..16 上电平变化触发中断请求时，PCIF2 置 "1”。如果 SREG 寄存器中的 I 位 I 与 PCICR 寄存器中的位 PCIE2 置 "1”，MCU 将会跳转到相应的中断向量。当中断程序执行时，该标志被清除。该位也可通过写逻辑 "1” 来 清除。

- **位 1 - PCIF1: 引脚电平变化中断标志 1**

当引脚 PCINT14..8 上电平变化触发中断请求时，PCIF1 置 "1”。如果 SREG 寄存器中的 I 位 I 与 PCICR 寄存器中的位 PCIE1 置 "1”，MCU 将会跳转到相应的中断向量。当中断程序执行时，该标志被清除。该位也可通过写逻辑 "1” 来 清除。

- **位 0 - PCIF0: 引脚电平变化中断标志 0**

当引脚 PCINT7..0 上电平变化触发中断请求时，PCIF0 置 "1”。如果 SREG 寄存器中的 I 位 I 与 PCICR 寄存器中的位 PCIE0 置 "1”，MCU 将会跳转到相应的中断向量。当中断程序执行时，该标志被清除。该位也可通过写逻辑 "1” 来 清除。

引脚电平变化屏蔽寄存器 2— PCMSK2

位	7	6	5	4	3	2	1	0	
	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..0 – PCINT23..16: 引脚电平变化使能屏蔽 23..16**

PCINT23..16中的每一位决定相应的I/O引脚电平变化中断是否使能。如果PCINT23..16与PCICR上的PCIE2位置位，则相应的引脚电平变化中断使能。如果PCINT23..16清零，相应的引脚电平变化中断禁用。

引脚电平变化屏蔽寄存器 1— PCMSK1

位	7	6	5	4	3	2	1	0	
	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
读 / 写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – Res: 保留位**

ATmega48/88/168 没有使用这些位，其读返回值为 "0”。

- **位 6..0 – PCINT14..8: 引脚电平变化使能屏蔽 14..8**

PCINT14..8 中的每一位决定相应的 I/O 引脚电平变化中断是否使能。如果 PCINT14..8 与 PCICR 上的 PCIE1 位置位，则相应的引脚电平变化中断使能。如果 PCINT14..8 清零，相应的引脚电平变化中断禁用。

引脚变化屏蔽寄存器 0— PCMSK0

位	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..0 – PCINT7..0: 引脚电平变化使能屏蔽 7..0**

PCINT7..0 中的每一位决定相应的 I/O 引脚电平变化中断是否使能。如果 PCINT7..0 与 PCICR 上的 PCIE0 位置位，则相应的引脚电平变化中断使能。如果 PCINT7..0 清零，相应引脚电平变化中断禁用。

具有 PWM 功能的 8 位定时器 / 计时器 0

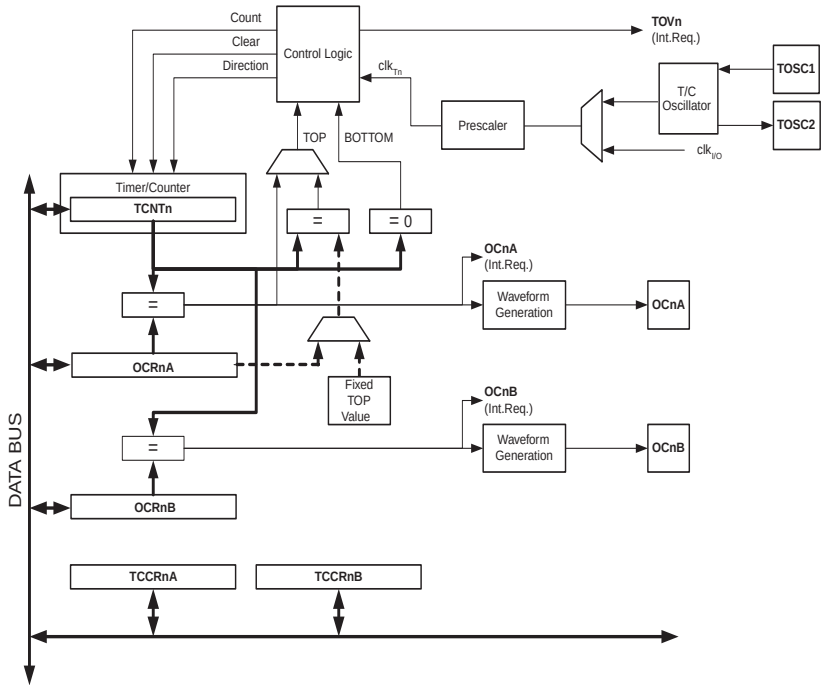
T/C0是一个通用的8位定时器/计数器模块，有两个独立的输出比较单元，且支持PWM功能。它提供精确的程序定时（事件管理）与波形产生。其主要特点如下：

- **两个独立的输出比较单元**
- **双缓冲输出比较寄存器**
- **比较匹配发生时清除定时器 (自动加载)**
- **无干扰脉冲，相位正确的 PWM**
- **可变 PWM 周期**
- **频率发生器**
- **三个独立的中断源 (TOV0, OCF0A 与 OCF0B)**

综述

Figure28为8位定时器/计数器的简化框图。实际引脚排列请参考P2”ATmega48/88/168引脚排列 ”。 CPU 可以访问的 I/O 寄存器，包括位和引脚，以粗体显示。 I/O 寄存器和位的位置列于 P89”8 位定时器 / 计数器寄存器的说明 ”。

Figure 28. 8 位 T/C 方框图



定义

本文的许多寄存器及其各个位以通用的格式表示。小写的 “n” 取代了 T/C 的序号，在此即为 0。小写的 “x” 取代了输出比较单元通道，在此即为通道 A 或通道 B。但是在写程序时要使用精确的格式，例如使用 TCNT0 来访问 T/C0 计数器值，等等。

Table45 的定义适用于全文。

Table 45. 定义

BOTTOM	计数器计到 0x00 时即达到 BOTTOM
MAX	计数器计到 0xFF (十进制的 255) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。TOP 值可以为固定值 0xFF (MAX)，或是存储于寄存器 OCR0A 里的数值，具体由工作模式确定

寄存器

T/C(TCNT0)和输出比较寄存器(OCR0A与OCR0B)为8位寄存器。中断请求(图中简写为 Int.Req.) 信号在定时器中断标志寄存器 TIFR0 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK0 单独进行屏蔽。图中没有给出 TIFR0 和 TIMSK0。

T/C 可以由内部时钟源通过预分频器驱动，或者通过T0 引脚的外部时钟源来驱动。时钟选择逻辑模块控制使用哪一个时钟源与什么边沿来增加 (或降低)T/C 的数值。如果没有选择时钟源 T/C 就不工作。时钟选择模块的输出定义为定时器时钟 clk_{T0}。

双缓冲的输出比较寄存器 (OCR0A 与 OCR0B) 持续地与 T/C 的数值进行比较。比较的结果可用来产生 PWM 波，或在输出比较引脚 (OCR0A 与 OCR0B) 上产生变化频率的输出，如 P105“ 使用输出比较单元 ” 说明的那样。比较匹配事件还将置位比较标志 (OCF0A 或 OCF0B)。此标志可以用来产生输出比较中断请求。

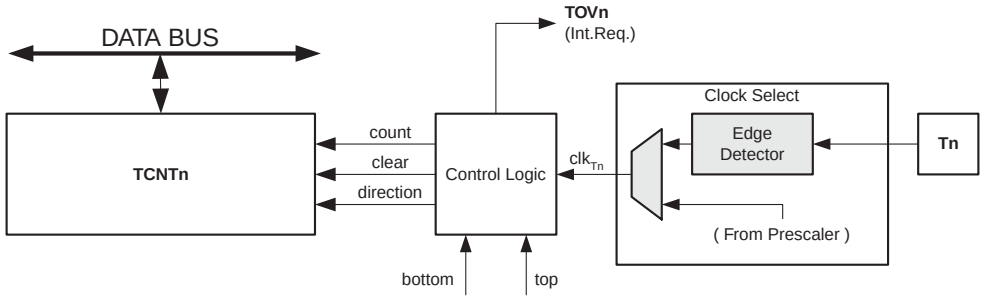
T/C 的时钟源

T/C 可以由内部同步时钟或外部异步时钟驱动。时钟源是由时钟选择逻辑决定的，而时钟选择逻辑是由位于 T/C 控制寄存器 TCCR0B 的时钟选择位 CS02:0 控制的。P95”T/C0 与 T/C1 的预分频器 ” 对时钟源与预分频有详尽的描述。

计数器单元

8 位 T/C 的主要部分为可编程的双向计数单元。Figure29 即为计数器和它周边电路的框图。

Figure 29. 计数器单元方框图



信号说明 (内部信号) :

count 使 TCNT0 加 1 或减 1

direction 选择加操作或减操作

clear 清除 TCNT0 (将所有的位清零)

clk_{Tn} T/C 的时钟，clk_{T0}

top 表示 TCNT0 已经达到了最大值

bottom 表示 TCNT0 已经达到了最小值 (0)

根据不同的工作模式，计数器针对每一个 clk_{T0} 实现清零、加一或减一操作。clk_{T0} 可以由内部时钟源或外部时钟源产生，具体由时钟选择位 CS02:0 确定。没有选择时钟源时 (CS02:0 = 0) 定时器即停止。但是不管有没有 clk_{T0}，CPU 都可以访问 TCNT0。CPU 写操作比计数器其他操作 (如清零、加减操作) 的优先级高。

计数序列由T/C控制寄存器 (TCCR0A)的WGM01和WGM00与T/C控制寄存器(TCCR0B)的 WGM02 决定。计数器计数行为与输出比较 OC0A 与 OC0B 的波形有紧密的关系。有关计数序列和波形产生的详细信息请参考 P84” 工作模式 ” 。

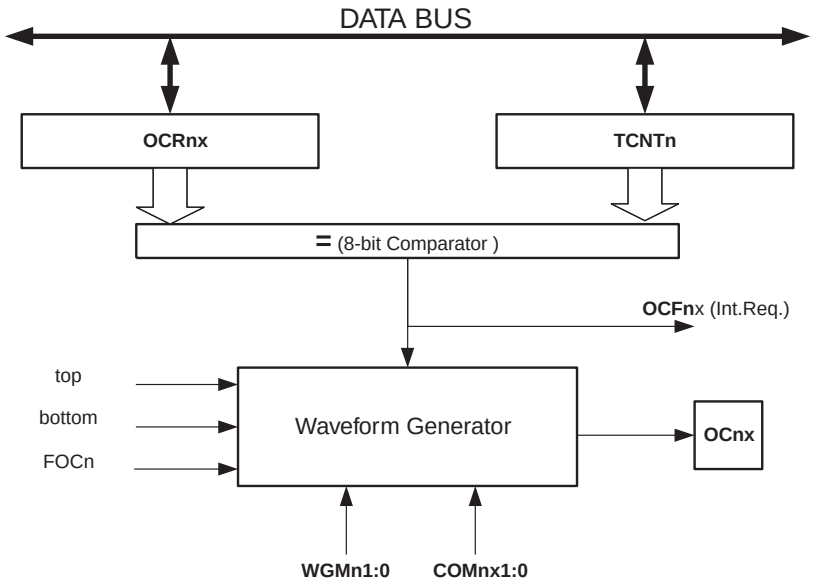
T/C溢出中断标志TOV0根据WGM02:0 设定的工作模式来设置。TOV0可以用于产生CPU中断。

输出比较单元

8位比较器持续对TCNT0和输出比较寄存器OCR0A与OCR0B进行比较。一旦TCNT0等于OCR0A 或 OCR0B ,比较器就给出匹配信号。在匹配发生的下一个定时器时钟周期输出比较标志 OCF0A 或 OCR0B 置位。若此时 OCIE0A = 1 且 SREG 的全局中断标志 I 置位，CPU 将产生输出比较中断。执行中断服务程序时OCF0A 自动清零 ,也可以通过软件写”1” 的方式来清零。根据由 WGM02:0 和 COM0x1:0 设定的不同的工作模式，波形发生器利用匹配信号产生不同的波形。同时，波形发生器还利用 max 和 bottom 信号来处理极值条件下的特殊情况 (P84” 工作模式 ”)。

Figure30 为输出比较单元的方框图。

Figure 30. 输出比较单元方框图



使用 PWM 模式时 OCR0x 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式双缓冲功能是禁止的。双缓冲可以将更新 OCR0x 寄存器与 top 或 bottom 时刻同步起来，从而防止产生不对称的 PWM 脉冲，消除了干扰脉冲。

访问 OCR0x 寄存器看起来很复杂 ,其实不然。使能双缓冲功能时 ,CPU 访问的是 OCR0x 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR0x 本身。

强制输出比较

工作于非 PWM 模式时，可以通过对强制输出比较位 FOC0x 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCF0x 标志，也不会重载 / 清零定时器，但是 OC0x 引脚将被更新，好象真的发生了比较匹配一样 (COM0x1:0 决定 OC0x 是置位、清零，还是 "0“-”1” 交替变化)。

写 TCNT0 操作将阻止比较匹配

CPU 对 TCNT0 寄存器的写操作会在下一个定时器时钟周期阻止比较匹配的发生 ,即使此时定时器已经停止了。这个特性可以用来将 OCR0x 初始化为与 TCNT0 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT0 都将在下一个定时器时钟周期阻止比较匹配，在使用输出比较时改变 TCNT0 就会有风险，不论 T/C 此时是否在运行与否。如果写入的 TCNT0 的数值等于 OCR0x，比较匹配就被丢失了，造成不正确的波形发生结果。类似地，在计数器进行降序计数时不要对 TCNT0 写入等于 BOTTOM 的数据。

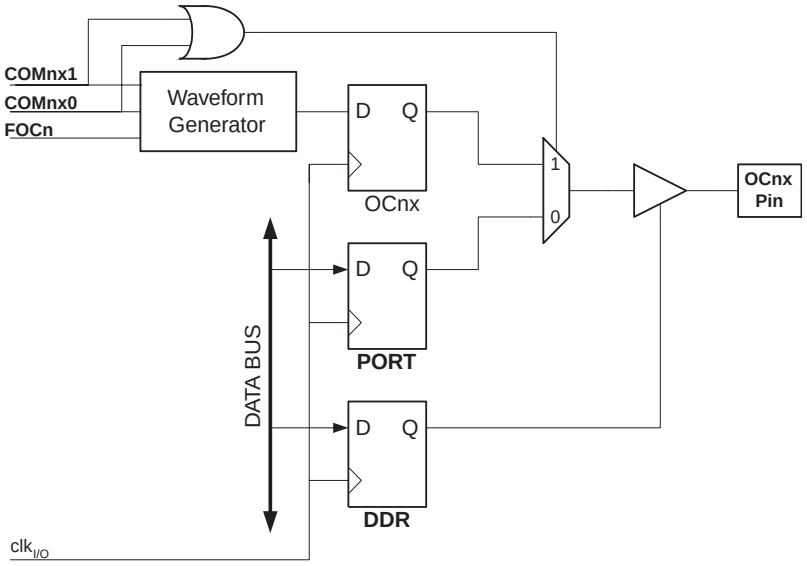
OC0x 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC0x 的方法是在普通模式下利用强制输出比较 FOC0x。即使在改变波形发生模式时 OC0x 寄存器也会一直保持它的数值。

注意 COM0x1:0 和比较数据都不是双缓冲的。 COM0x1:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM0x1:0 具有双重功能。波形发生器利用 COM0x1:0 来确定下一次比较匹配发生时的输出比较状态 (OC0x) ； COM0x1:0 还控制 OC0x 引脚输出信号的来源。 Figure31 为受 COM0x1:0 设置影响的简化逻辑框图。 I/O 寄存器、 I/O 位和 I/O 引脚以粗体表示。图中只给出了受 COM0x1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及OC0x状态时指的是内部OC0x寄存器，而不是OC0x引脚。系统复位时OC0x寄存器清零。

Figure 31. 比较匹配输出单元原理图



如果COM0x1:0不全为零，通用I/O口功能将被波形发生器的输出比较功能取代。但OC0x引脚为输入还是输出仍然由数据方向寄存器 DDR 控制。在使用 OC0x 功能之前首先要通过数据方向寄存器的 DDR_OC0x 位将此引脚设置为输出。端口功能与波形发生器的工作模式无关。

输出比较逻辑的设计允许 OC0x 状态在输出之前首先进行初始化。要注意某些 COM0x1:0 设置保留给了其他特定操作，详见 P89“8 位定时器 / 计数器寄存器的说明 ”。

比较输出模式和波形产生

波形发生器利用 COM0x1:0 的方法在普通模式、 CTC 模式和 PWM 模式下有所区别。对于所有的模式，设置 COM0x1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC0x 寄存器。非PWM模式的比较输出请参见P88Table46 ；快速PWM的比较输出示于P88Table47 ；相位修正 PWM 的比较输出在 P88Table48 有描述。

改变 COM0x1:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC0x 来立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM02:0) 及比较输出模式 (COM0x1:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM0x1:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COM0x1:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P83“ 比较匹配输出单元 ”)。

具体的时序信息请参考 P87”T/C 时序图 ” 。

普通模式

普通模式 (WGM02:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到 8 比特的最大值后 (TOP = 0xFF)，由于数值溢出计数器简单地返回到最小值 0x00 重新开始。在 TCNT0 为零的同一个定时器时钟里 T/C 溢出标志 TOV0 置位。此时 TOV0 有点象第 9 位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV0，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

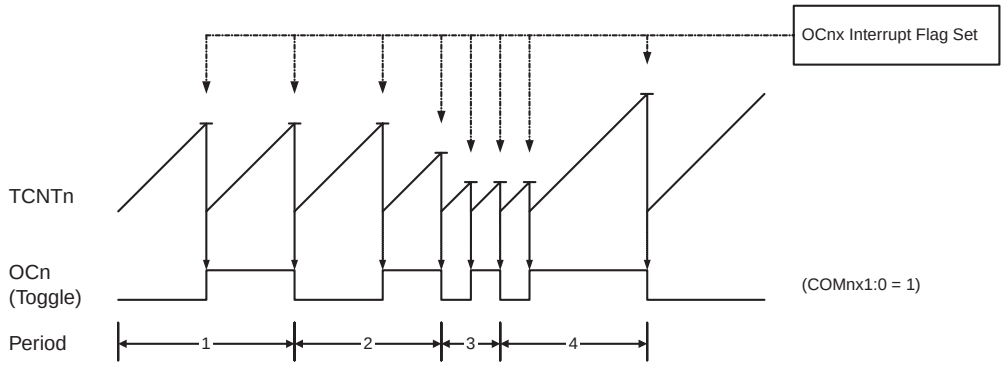
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形，因为这会占用太多的 CPU 时间。

CTC(比较匹配时清零定时器) 模式

在 CTC 模式 (WGM02:0 = 2) 下 OCR0A 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT0 等于 OCR0A 时计数器清零。OCR0A 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为 Figure32。计数器数值TCNT0一直累加到TCNT0与OCR0A匹配，然后 TCNT0 清零。

Figure 32. CTC 模式的时序图



利用 OCF0A 标志可以在计数器数值达到 TOP 时产生中断。在中断服务程序里可以更新 TOP的数值。由于CTC模式没有双缓冲功能，在计数器以无预分频器或很低的分频系数工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入的 OCR0A 数值小于当前 TCNT0 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFF，然后再从 0x00 开始计数到 OCF0A。

为了在CTC模式下得到波形输出，可以设置OC0A在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM0A1:0 = 1 来完成。在期望获得 OC0A 输出之前，首先要将其端口设置为输出。波形发生器能够产生的最大频率为 $f_{OC0} = f_{clk_I/O}/2$ (OCR0A = 0x00)。频率由如下公式确定：

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

变量 N 代表预分频因子 (1、 8、 64、或 1024)。

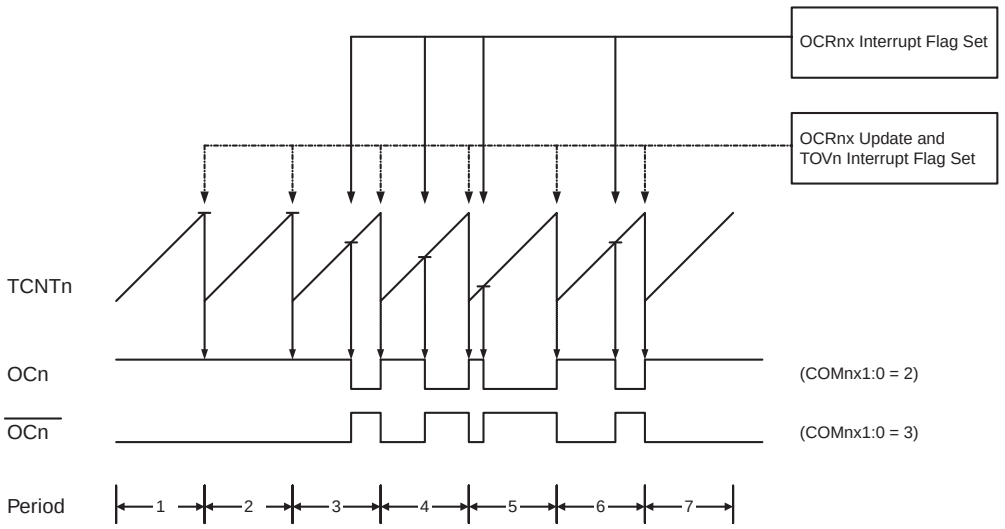
在普通模式下， TOV0 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

快速 PWM 模式

快速 PWM 模式 (WGM02:0 = 3 或 7) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单斜坡工作方式。计数器从 BOTTOM 计到 TOP，然后立即回到 BOTTOM 重新开始。当 WGM2:0 = 3 时，TOP 定义为 0xFF，而当 WGM2:0 = 7 时，TOP 定义为 OCR0A 。对于普通的比较输出模式，输出比较引脚 OC0x 在 TCNT0 与 OCR0x 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式，OC0x 的动作正好相反。由于使用了单斜坡模式，快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，计数器的数值一直增加到 MAX，然后在紧接的时钟周期清零。具体的时序图为 Figure33。图中柱状的 TCNT0 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。 TCNT0 斜坡上的短水平线表示 OCR0x 和 TCNT0 的比较匹配。

Figure 33. 快速 PWM 模式时序图



计时器数值达到 TOP 时 T/C 溢出标志 TOV0 置位。如果中断使能，在中断服务程序可以更新比较值。

工作于快速 PWM 模式时，比较单元可以在 OC0x 引脚上输出 PWM 波形。COM0x1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形：如果 WGM02 位置位，将 COM0A1:0设置为"1"可以使OC0A引脚在比较匹配时电平交替变化。OC0B引脚没有这个功能 (参见 P89Table50)。要想在引脚上得到输出信号还必须将 OC0x 的数据方向设置为输出。产生 PWM 波形的机理是 OC0x 寄存器在 OCR0x 与 TCNT0 匹配时置位 (或清零)，以及在计数器清零 (从 TOP 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

变量 N 代表分频因子 (1、8、64、256 或 1024)。

OCR0A 寄存器为极限值时表示快速 PWM 模式的一些特殊情况。若 OCR0A 等于 BOTTOM，输出为出现在第MAX+1个定时器时钟周期的窄脉冲；OCR0A为MAX时，根据 COM0A1:0 的设定，输出恒为高电平或低电平。

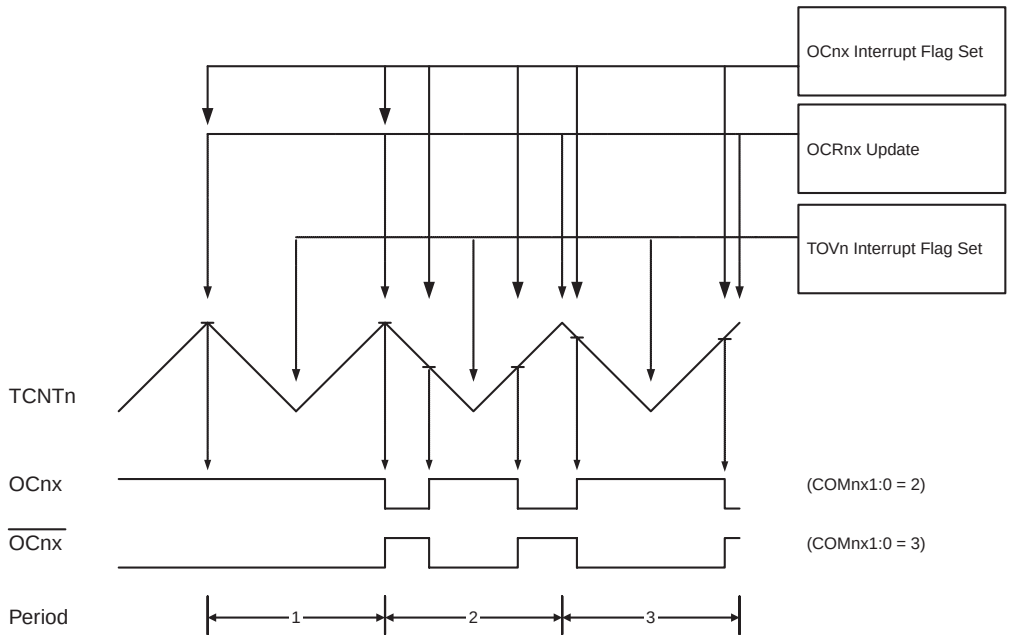
通过设定 OC0A 在比较匹配时进行逻辑电平取反 (COM0x1:0 = 1)，可以得到占空比为 50%的周期信号。OCR0A为0时信号有最高频率$f_{oc2} = f_{clk_I/O}/2$。这个特性类似于CTC模式下的 OC0A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

相位修正 PWM 模式 (WGM02:0 = 1 或 5) 为用户提供了一个获得高精度相位修正 PWM 波形的方法。此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。当 WGM2:0 = 1 时，TOP 定义为 0xFF，而当 WGM2:0 = 5 时，TOP定义为OCR0A。在一般的比较输出模式下，当计时器往TOP计数时若发生了TCNT0与 OCR0x 的匹配， OC0x 将清零为低电平；而在计时器往 BOTTOM 计数时若发生了TCNT0与OCR0x的匹配，OC0x将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但由于其对称的特性，十分适合于电机控制。

相位修正 PWM 模式下，计时器不断地累加直到 TOP，然后开始减计数。在一个定时器时钟周期里 TCNT0 的值等于 TOP。时序图可参见 Figure34。图中 TCNT0 的数值用柱状图表示，以说明双斜坡操作。本图同时说明了普通 PWM 的输出和反向 PWM 的输出。TCNT0 斜坡上的小横条表示 OCR0x 与 TCNT0 的比较匹配。

Figure 34. 相位修正 PWM 模式的时序图



当计时器达到 BOTTOM 时 T/C 溢出标志位 TOV0 置位。此标志位可用来产生中断。

工作于相位修正 PWM 模式时，比较单元可以在 OC0x 引脚产生 PWM 波形：将 COM0x1:0 设置为 2 产生普通相位的 PWM，设置 COM0x1:0 为 3 产生反向 PWM 信号：如果 WGM02 位置位，将 COM0A 设置为 "1" 可以使 OC0A 引脚在比较匹配时输出电平取反。该选项对OC0B引脚无效 (参见P89Table51)。要想在引脚上得到输出信号还必须将OC0x的数据方向设置为输出。 OCR0x 和 TCNT0 比较匹配发生时 OC0x 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由下式公式获得：

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

变量 N 表示预分频因子 (1、8、64、 256 或 1024)。

OCR0A 寄存器处于极值代表了相位修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR0A 等于 BOTTOM，输出一直保持为低电平；若 OCR0A 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

在 Figure34 的第 2 个周期，虽然没有发生比较匹配， OCnx 也出现了一个从高到低的跳变。其目的是保证波形在BOTTOM两侧的对称。没有比较匹配时有两种情况会出现跳变。

- 如 Figure34 所示，OCRnx 的值从 MAX 改变为其他数据。当 OCR0A 值为 MAX 时，引脚 OCn 的输出应该与前面降序记数比较匹配的结果相同。为了保证波形在 BOTTOM 两侧的对称，当 T/C 的数值为 MAX 时，引脚 OCnx 的输出又必须符合后面升序记数比较匹配的结果。此时就出现了虽然没有比较匹配发生 OCnx 却仍然有跳变的现象。
- 定时器从一个比 OCRnx 高的值开始记数，并因而丢失了一次比较匹配。为了保证波形在 BOTTOM 两侧的对称，此时 OCnx 必须立即切换到合适的电平。系统因此引入了第二个虽然没有比较匹配发生 OCnx 却仍然有跳变的现象。

T/C 时序图

T/C 是同步电路，因此其时钟 clk_{T0} 可以表示为时钟使能信号，如下图所示。图中还说明了中断标志设置的时间。 Figure35 给出了基本的 T/C 工作时序，以及除了相位修正 PWM 模式之外其他模式接近 MAX 时的记数序列。

Figure 35. T/C 时序图，无预分频器

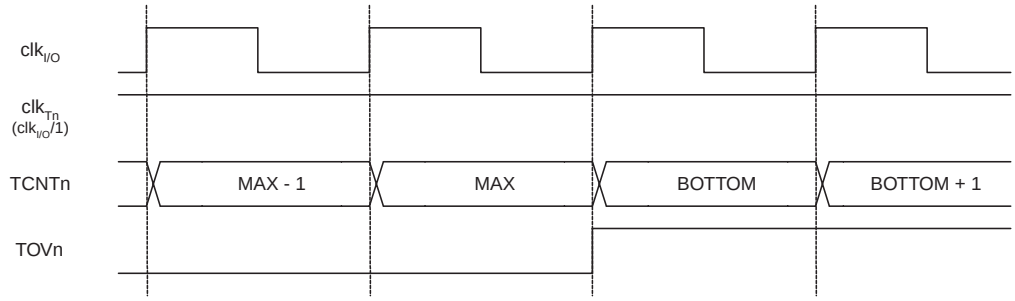


Figure36 所示为相同的工作时序，但有预分频。

Figure 36. T/C 时序图，预分频器为 $f_{clk_I/O}$/8

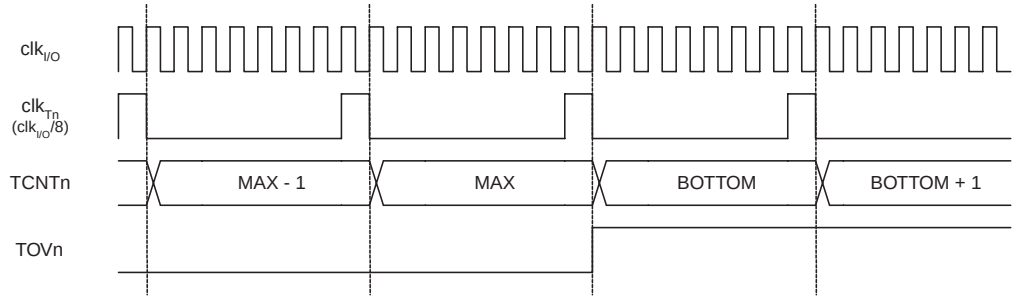


Figure37给出了各种模式下OCF0B，以及除了CTC 模式和PWM模式下OCF0A的置位情况。 OCR0A 的值为 TOP。

Figure 37. T/C 时序图， OCF0x 置位，预分频器为 $f_{clk_I/O}/8$

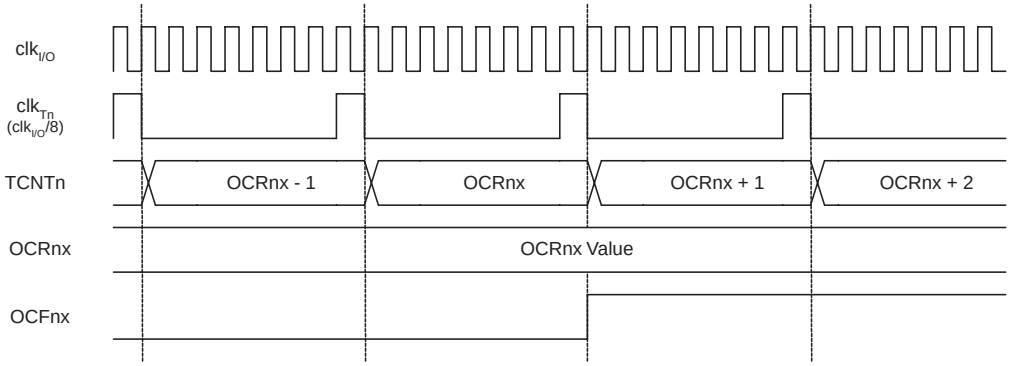
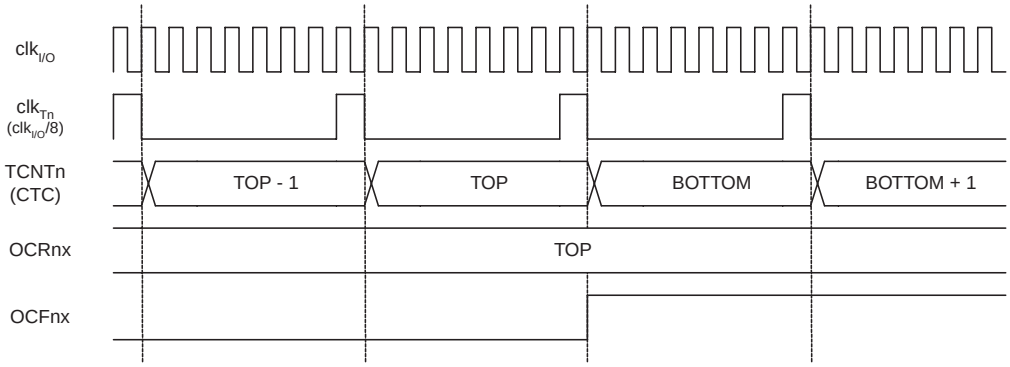


Figure38 给出了CTC模式与快速PWM模式下OCF0A置位和TCNT0清除的情况。OCR0A的值为 TOP。

Figure 38. T/C 时序图， CTC 模式，预分频器为 $f_{clk_I/O}/8$



T/C 控制寄存器 A—TCCR0A

位	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
读 / 写	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7:6 – COM0A1:0: 比较匹配输出 A 模式**

这些位决定了比较匹配发生时输出引脚 OC0A 的电平。如果 COM0A1:0 中的一位或全部置位， OC0A 以比较匹配输出的方式进行工作。同时其方向控制位要设置为 1 以使能输出驱动器。

当OC0A连接到物理引脚上时，COM0A1:0位的功能由WGM02:0位的设置决定。Table46给出当 WGM02:0 设置位普通或 CTC 模式 (非 PWM 模式) 时， COM0A1:0 位的功能。

Table 46. 比较输出模式，非 PWM 模式

COM0A1	COM0A0	说明
0	0	正常的端口操作，不与 OC0A 相连接
0	1	比较匹配发生时 OC0A 取反
1	0	比较匹配发生时 OC0A 清零
1	1	比较匹配发生时 OC0A 置位

Table47 给出了当 WGM01:0 设置为快速 PWM 模式时 COM0A1:0 的功能。

Table 47. 比较输出模式，快速 PWM 模式 (1)

COM0A1	COM0A0	说明
0	0	正常的端口操作，不与 OC0A 相连接
0	1	WGM02 = 0: 正常的端口操作，不与 OC0A 相连接 WGM02 = 1: 比较匹配发生时 OC0A 取反
1	0	比较匹配发生时 OC0A 清零，计数到 TOP 时 OC0A 置位
1	1	比较匹配发生时 OC0A 置位，计数到 TOP 时 OC0A 清零

Note: 1. 一个特殊情况是OCR0A等于TOP，且COM0A1置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0A 的动作继续有效。详细信息请参见 P85" 快速 PWM 模式 " 。

Table48 给出了当 WGM02:0 设置为相位修正 PWM 模式时 COM0A1:0 的功能。

Table 48. 比较输出模式，相位修正 PWM 模式 (1)

COM0A1	COM0A0	说明
0	0	正常的端口操作，不与 OC0A 相连接
0	1	WGM02 = 0: 正常的端口操作，不与 OC0A 相连接 WGM02 = 1: 比较匹配发生时 OC0A 取反
1	0	在升序计数时发生比较匹配将清零 OC0A ；降序计数时发生比较匹配将置位 OC0A
1	1	在升序计数时发生比较匹配将置位 OC0A ；降序计数时发生比较匹配将清零 OC0A

Note: 1. 一个特殊情况是OCR0A等于TOP，且COM0A1置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0A 的动作继续有效。详细信息请参见 P111” 相位修正 PWM 模式 ” 。

- **位 5:4 – COM0B1:0: 比较匹配输出 B 模式**

这些位决定了比较匹配发生时输出引脚 OC0B 的电平。如果 COM0B1:0 中的一位或全部置位， OC0B 以比较匹配输出的方式进行工作。同时其方向控制位要设置为 1 以使能输出驱动器。

当 OC0B 连接到物理引脚上时， COM0B1:0 的功能依赖于 WGM02:0 的设置。 Table49 给出了当 WGM02:0 设置为普通模式或 CTC 模式时 COM0B1:0 的功能。

Table 49. 比较输出模式，非 PWM 模式

COM0B1	COM0B0	说明
0	0	正常的端口操作，不与 OC0B 相连接
0	1	比较匹配发生时 OC0B 取反
1	0	比较匹配发生时 OC0B 清零
1	1	比较匹配发生时 OC0B 置位

Table50 给出了当 WGM02:0 设置为快速 PWM 模式时 COM0B1:0 的功能。

Table 50. 比较输出模式，快速 PWM 模式 (1)

COM0B1	COM0B0	说明
0	0	正常的端口操作，不与 OC0B 相连接
0	1	保留
1	0	比较匹配发生时 OC0B 清零，计数到 TOP 时 OC0B 置位
1	1	比较匹配发生时 OC0B 置位，计数到 TOP 时 OC0B 清零

Note: 1. 一个特殊情况是OCR0B等于TOP，且COM0B1置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0B 的动作继续有效。详细信息请参见 P85” 快速 PWM 模式 ” 。

Table51 给出了当 WGM02:0 设置为相位修正 PWM 模式时 COM0B1:0 的功能。

Table 51. 比较输出模式，相位修正 PWM 模式 (1)

COM0B1	COM0B0	说明
0	0	正常的端口操作，不与 OC0B 相连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC0B ；降序计数时发生比较匹配将置位 OC0B
1	1	在升序计数时发生比较匹配将置位 OC0B ；降序计数时发生比较匹配将清零 OC0B

Note: 1. 一个特殊情况是OCR0B等于TOP，且COM0B1置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0B 的动作继续有效。详细信息请参见 P86” 相位修正 PWM 模式 ” 。

- **位 3, 2 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 "0”。

- **位 1:0 – WGM01:0: 波形产生模式**

这几位与 TCCR0B 寄存器的 WGM02 结合起来控制计数器的计数序列，计数器的最大值 TOP，以及产生何种波形，详见 Table52。T/C 支持的模式有：普通模式，比较匹配发生时清除计数器模式 (CTC)，以及两种 PWM 模式 (见 P84" 工作模式 ")。

Table 52. 波形产生模式的位定义

模式	WGM02	WGM01	WGM00	T/C 的工作模式	TOP	OCRx 的更新时间	TOV 的置位时刻 (1)(2)
0	0	0	0	普通	0xFF	立即更新	MAX
1	0	0	1	PWM，相位修正	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	立即更新	MAX
3	0	1	1	快速 PWM	0xFF	TOP	MAX
4	1	0	0	保留	–	–	–
5	1	0	1	PWM，相位修正	OCRA	TOP	BOTTOM
6	1	1	0	保留	–	–	–
7	1	1	1	快速 PWM	OCRA	TOP	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

T/C 控制寄存器 B—TCCR0B

位	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
读 / 写	W	W	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – FOC0A: 强制输出比较 A**

FOC0A 仅在 WGM 指明非 PWM 模式时才有效。

但是，为了保证与未来器件的兼容性，在使用 PWM 时，写 TCCR0B 要对其清零。对其写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC0A 将按照 COM0A1:0 的设置输出相应的电平。要注意 FOC0A 类似一个锁存信号，真正对强制输出比较起作用的是 COM0A1:0 的设置。

FOC0A不会引发任何中断，也不会在利用OCR0A作为TOP的CTC模式下对定时器进行清零的操作。

读 FOC0A 的返回值永远为 0。

- **位 6 – FOC0B: 强制输出比较 B**

FOC0B 仅在 WGM 指明非 PWM 模式时才有效。

但是，为了保证与未来器件的兼容性，在使用 PWM 时，写 TCCR0B 要对其清零。对其写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC0B 将按照 COM0B1:0 的设置输出相应的电平。要注意 FOC0B 类似一个锁存信号，真正对强制输出比较起作用的是 COM0B1:0 的设置。

FOC0B不会引发任何中断，也不会在利用OCR0B作为TOP的CTC模式下对定时器进行清零的操作。

读 FOC0B 的返回值永远为 0。

- **位 5:4 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 "0”。

- **位 3 – WGM02: 波形产生模式**

请参见 P89”T/C 控制寄存器 A – TCCR0A” 。

- **位 2:0 – CS02:0: 时钟选择**

用于选择 T/C 的时钟源。

Table 53. 时钟选择位定义

CS02	CS01	CS00	说明
0	0	0	无时钟， T/C 不工作
0	0	1	$clk_{I/O}$/1 (没有预分频)
0	1	0	$clk_{I/O}$/8 (来自预分频器)
0	1	1	$clk_{I/O}$/64 (来自预分频器)
1	0	0	$clk_{I/O}$/256 (来自预分频器)
1	0	1	$clk_{I/O}$/1024 (来自预分频器)
1	1	0	时钟由 T0 引脚输入，下降沿触发
1	1	1	时钟由 T0 引脚输入，上升沿触发

如果 T/C0 使用外部时钟，即使 T0 被配置为输出，其上的电平变化仍然会驱动记数器。利用这一特性可通过软件控制记数。

T/C 寄存器—TCNT0

位	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT0 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT0 的数值有可能丢失一次 TCNT0 和 OCR0x 的比较匹配。

输出比较寄存器 A—OCR0A

位	7	6	5	4	3	2	1	0	
	OCR0A[7:0]								OCR0A
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 A 包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0A 引脚上产生波形。

输出比较寄存器 B—OCR0B

位	7	6	5	4	3	2	1	0	
	OCR0B[7:0]								OCR0B
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 B 包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0B 引脚上产生波形。

T/C 中断屏蔽寄存器—TIMSK0

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
读 / 写	R	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..3 – Res: 保留位**

ATmega48/88/168 中的这些位保留，读返回值为 0。

- **位 2 – OCIE0B: T/C 输出比较匹配 B 中断使能**

当 OCIE0B 和状态寄存器的全局中断使能位 I 都为 ”1” 时，T/C 的输出比较匹配 B 中断使能。当 T/C 的比较匹配发生，即 TIFR0 中的 OCF0B 置位时，中断服务程序得以执行。

- **位 1 – OCIE0A: T/C0 输出比较匹配 A 中断使能**

当 OCIE0A 和状态寄存器的全局中断使能位 I 都为 ”1” 时， T/C0 的输出比较匹配 A 中断使能。当 T/C0 的比较匹配发生 ,即 TIFR0 中的 OCF0A 置位时 ,中断服务程序得以执行。

- **位 0 – TOIE0: T/C0 溢出中断使能**

当 TOIE0 和状态寄存器的全局中断使能位 I 都为 ”1” 时，T/C0 的溢出中断使能。当 T/C0 发生溢出，即 TIFR0 中的 TOV0 位置位时，中断服务程序得以执行。

T/C 0 中断标志寄存器—TIFR0

位	7	6	5	4	3	2	1	0

	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
读 / 写	R	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..3 – Res: 保留**

ATmega48/88/168 中的这些位保留，读返回值为 0。

- **位 2 – OCF0B: T/C0 输出比较 B 匹配标志**

当 T/C 与 OCR0B(输出比较寄存器 0B) 的值匹配时， OCF0B 置位。此位在中断服务程序里硬件清零，也可以对其写 1 来清零。当 SREG 中的位 I、OCIE0B(T/C 比较 B 匹配中断使能) 和 OCF0B 都置位时，中断服务程序得到执行。

- **位 1 – OCF0A: T/C0 输出比较 A 匹配标志**

当 T/C0 与 OCR0A(输出比较寄存器 0A) 的值匹配时， OCF0A 置位。此位在中断服务程序里硬件清零，也可以对其写 1 来清零。当 SREG 中的位 I、 OCIE0A(T/C0 比较匹配中断使能) 和 OCF0A 都置位时，中断服务程序得到执行。

- **位 0 – TOV0: T/C0 溢出标志**

T/C0 溢出时 TOV0 置位。执行相应的中断服务程序时此位硬件清零。此外 ,TOV0 也可以通过写 1 来清零。当 SREG 中的位 I、 TOIE0(T/C0 溢出中断使能) 和 TOV0 都置位时，中断服务程序得到执行。

该标志位的设置取决于 WGM02:0 位的设定。请参见 Table52，P90” 波形产生模式的位定义 ”。

T/C0 与 T/C1 的预分频器

T/C1 与 T/C0 共用一个预分频模块，但它们可以有不同的分频设置。下述内容适用于 T/C1 与 T/C0。

内部时钟源

当 CSn2:0 = 1 时，系统内部时钟直接作为 T/C 的时钟源，这也是 T/C 最高频率的时钟源 $f_{CLK_I/O}$，与系统时钟频率相同。预分频器可以输出 4 个不同的时钟信号 $f_{CLK_I/O}/8$、$f_{CLK_I/O}/64$、$f_{CLK_I/O}/256$ 或 $f_{CLK_I/O}/1024$。

分频器复位

预分频器是独立运行的。也就是说，其操作独立于 T/C 的时钟选择逻辑，且它由 T/C1 与 T/C0 共享。由于预分频器不受 T/C 时钟选择的影响，预分频器的状态需要包含预分频时钟被用到何处这样的信息。一个典型的例子发生在定时器使能并由预分频器驱动 (6 > CSn2:0 > 1)的时候：从计时器使能到第一次开始计数可能花费 1 到N+1个系统时钟周期，其中 N 等于预分频因子 (8、 64、 256 或 1024)。

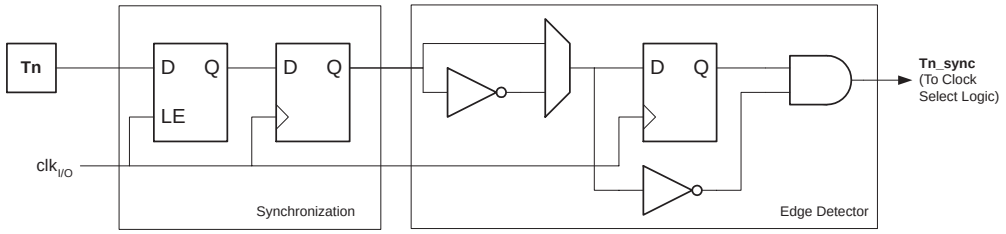
通过复位预分频器来同步 T/C 与程序运行是可能的。但是必须注意另一个 T/C 是否也在使用这一预分频器，因为预分频器复位将会影响所有与其连接的 T/C。

外部时钟源

由 T1/T0 引脚提供的外部时钟源可以用作 T/C 时钟 clk_{T1}/clk_{T0}。引脚同步逻辑在每个系统时钟周期对引脚 T1/T0 进行采样。然后将同步 (采样) 信号送到边沿检测器。Figure39 给出了 T1/T0 同步采样与边沿检测逻辑的功能等效方框图。寄存器由内部系统时钟 $clk_{I/O}$ 的上跳沿驱动。当内部时钟为高时，锁存器可以看作时透明的。

CSn2:0 = 7 时边沿检测器检测到正跳变后产生 clk_{T1} 脉冲； CSn2:0 = 6 时负跳变引发 clk_{T0} 脉冲。

Figure 39. T1/T0 引脚采样



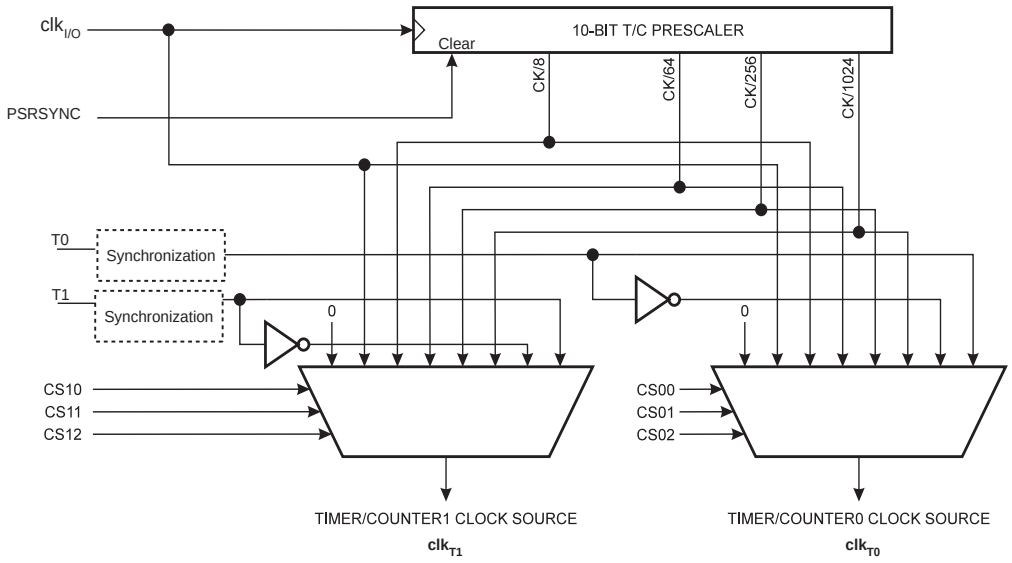
由于引脚上同步与边沿监测电路的存在，引脚 T1/T0 上的电平变化需要延时 2.5 到 3.5 个系统时钟周期才能使计数器进行更新。

禁止或使能时钟输入必须在 T1/T0 保持稳定至少一个系统时钟周期后才能进行，否则有产生错误 T/C 时钟脉冲的危险。

为保证正确的采样，外部时钟脉冲宽度必须大于一个系统时钟周期。在占空比为 50% 时外部时钟频率必须小于系统时钟频率的一半 ($f_{ExtClk} < f_{clk_I/O}/2$)。由于边沿检测器使用的是采样这一方法，它能检测到的外部时钟最多是其采样频率的一半 (Nyquist 采样定理)。然而，由于振荡器 (晶体、谐振器与电容) 本身误差带来的系统时钟频率及占空比的差异，建议外部时钟的最高频率不要大于 $f_{clk_I/O}/2.5$。

外部时钟源不送入预分频器。

Figure 40. T/C0 与 T/C1 预分频器 (1)



Note: 1. 输入引脚 (T1/T0) 的同步逻辑见 Figure39.

通用 T/C 控制寄存器—GTCCR

位	7	6	5	4	3	2	1	0	
	TSM	–	–	–	–	–	PSRASY	PSRSYNC	GTCCR
读 / 写	R/W	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – TSM: T/C 同步模式**

TSM置位激活T/C同步模式。只要TSM置位 ,PSR2 与PSR10的数值将保持不变 ,使得相关的定时器 / 计数器预分频器处于持续复位状态。这样相关的 T/C 将停止工作。用户可以为它们赋予相同的数值而不会出现在配置一个定时器 / 计数器时另一个 T/C 在运行的现象。一旦TSM清零 ,PSR2 与PSR10位被硬件清零 ,相关的定时器/计数器同时开始计数。

- **位 0 – PSRSYNC: 预分频器复位**

PSRSYNC置位时T/C1与T/C0的预分频器复位。操作完成后这一位通常由硬件立即清零 ,除非TSM置位。要注意的是T/C1与T/C0共用一个预分频器 ,复位对两个计时器都有影响。

16 位定时器 / 计数器 1

16位的T/C可以实现精确的程序定时(事件管理)、波形产生和信号测量。其主要特点如下:

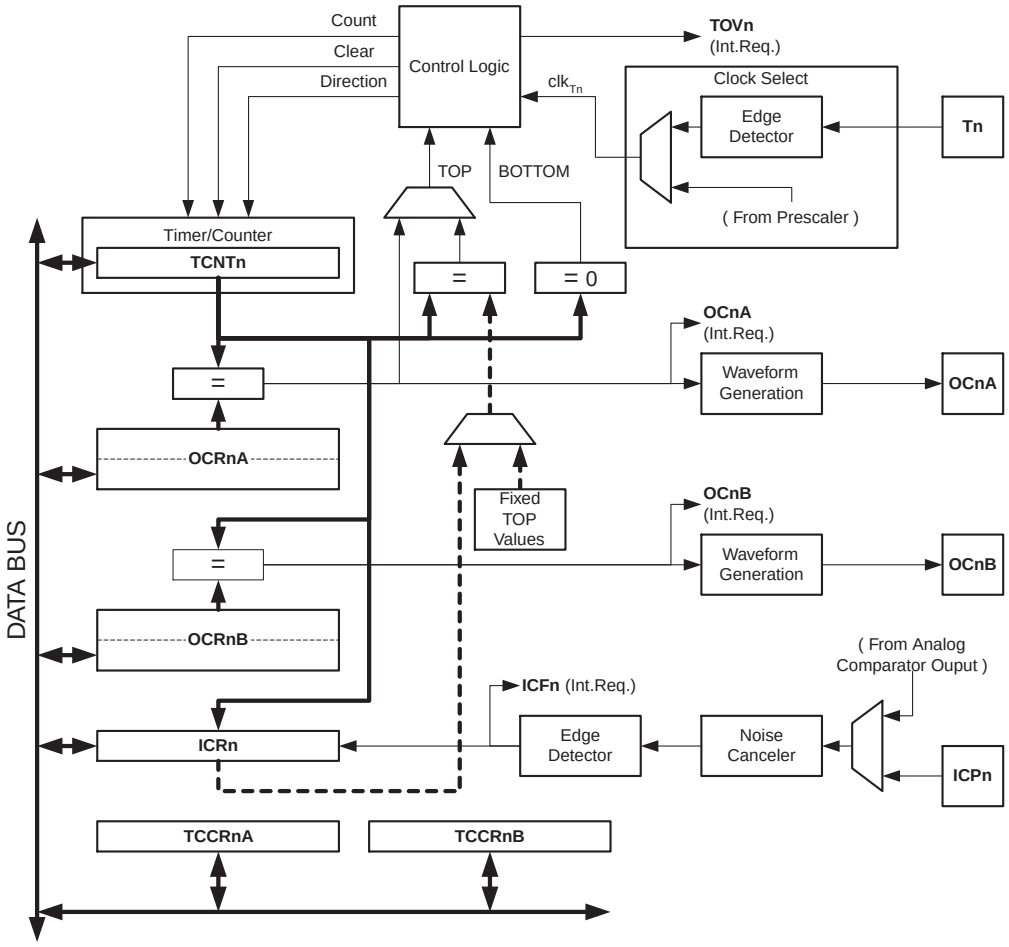
- **真正的 16 位设计 (即允许 16 位的 PWM)**
- **2 个独立的输出比较单元**
- **双缓冲的输出比较寄存器**
- **一个输入捕捉单元**
- **输入捕捉噪声抑制器**
- **比较匹配发生时清除寄存器 (自动重载)**
- **无干扰脉冲，相位正确的 PWM**
- **可变的 PWM 周期**
- **频率发生器**
- **外部事件计数器**
- **4 个独立的中断源 (TOV1、OCF1A、 OCF1B 与 ICF1)**

综述

本节大多数的寄存器和位定义以通用的方式表示。小写 “n” 表示 T/C 序号，小写 “x” 表示输出比较通道号。但是在写程序时要用完整的、精确的名称。如用 TCNT1 表示访问 T/C1 计数器值等。

16 位 T/C 的简化框图示于 Figure41。I/O 引脚的实际位置请参见 P2”ATmega48/88/168 引脚排列 ”。CPU 可访问的 I/O 寄存器，包括 I/O 位和 I/O 引脚以粗体表示。器件具体 I/O 寄存器与位定位见 P117”16 位定时器 / 计数器寄存器的说明 ”。

Figure 41. 16 位 T/C 框图 (1)



Note: 1. 请参考 P2Figure 1 ， P65Table34 和 P71Table40 以获得 T/C1 的引脚定义。

寄存器

定时器 / 计数器 TCNT1、输出比较寄存器 OCR1A/B 与输入捕捉寄存器 ICR1 均为 16 位寄存器。访问 16 位寄存器必须使用特殊的步骤，详见 P99" 访问 16 位寄存器 " 。 T/C 控制寄存器 TCCR1A/B 为 8 位寄存器，没有 CPU 访问的限制。中断请求 (图中简写为 Int.Req.) 信号在中断标志寄存器 TIFR1 都有反映。所有中断都可以由中断屏蔽寄存器 TIMSK1 单独控制。图中未给出 TIFR1 与 TIMSK1。

T/C可由内部时钟通过预分频器或通过由T1引脚输入的外部时钟驱动。引发T/C数值增加(或减少) 的时钟源及其有效沿由时钟选择逻辑模块控制。没有选择时钟源时 T/C 处于停止状态。时钟选择逻辑模块的输出称为 clk_{T1}。

双缓冲输出比较寄存器 OCR1A/B 持续地与 T/C 的值进行比较。波形发生器用比较结果产生 PWM 或在输出比较引脚 OC1A/B 输出可变频率的信号。参见 P104" 输出比较单元 " 。比较匹配结果还可置位比较匹配标志 OCF1A/B，用来产生输出比较中断请求。

当输入捕捉引脚 ICP1 或模拟比较器输入引脚 (见 P214" 模拟比较器 ") 有输入捕捉事件产生 (边沿触发) 时，当时的 T/C 值被传输到输入捕捉寄存器保存起来。输入捕捉单元包括一个数字滤波单元 (噪声消除器) 以降低噪声干扰。

在某些操作模式下， TOP 值或 T/C 的最大值可由 OCR1A 寄存器、 ICR1 寄存器，或一些固定数据来定义。在 PWM 模式下用 OCR1A 作为 TOP 值时， OCR1A 寄存器不能用作 PWM 输出。但此时 OCR1A 是双向缓冲的， TOP 值可在运行过程中得到改变。当需要一个固定的 TOP 值时可以使用 ICR1 寄存器 ,从而释放 OCR1A 来用作 PWM 的输出。

定义

以下定义适用于本节：

Table 54. 定义

BOTTOM	计数器计到 0x0000 时即达到 BOTTOM
MAX	计数器计到 0xFFFF (十进制的 65535) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。 TOP 值可以为固定值 0x00FF、0x01FF 或 0x03FF ,或是存储于寄存器 OCR1A 或ICR1 里的数值 ,具体数据依赖于工作模式

访问 16 位寄存器

TCNT1、OCR1A/B 与 ICR1 是 AVR CPU 通过 8 位数据总线进行访问的 16 位寄存器。读写 16 位寄存器需要两次操作。每个 16 位计时器都有一个 8 位临时寄存器用来存放高 8 位数据。每个 16 位定时器所属的 16 位寄存器共用相同的临时寄存器。访问低字节会触发 16 位读或写操作。当 CPU 写入数据到 16 位寄存器的低字节时，写入的 8 位数据与存放在临时寄存器中的高 8 位数据组成一个 16 位数据，同步写入到 16 位寄存器中。当 CPU 读取 16 位寄存器的低字节时，高字节内容在读低字节操作的同时被放置于临时辅助寄存器中。

并非所有的 16 位访问都涉及临时寄存器。对 OCR1A/B 寄存器的读操作就不涉及临时寄存器。

写 16 位寄存器时，应先写入该寄存器的高位字节。而读 16 位寄存器时应先读取该寄存器的低位字节。

下面的例程说明了如何访问 16 位定时器寄存器。前提是假设不会发生更新临时寄存器内容的中断。同样的原则也适用于对 OCR1A/B 与 ICR1 寄存器的访问。使用 “C” 语言时，编译器会自动处理 16 位操作。

汇编代码例程 (1)

```
...
; 设置 TCNT1 为 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; 将 TCNT1 读入 r17:r16
in  r16,TCNT1L
in  r17,TCNT1H
...
```

C 代码例程 (1)

```
unsigned int i;
...
/* 设置 TCNT1 为 0x01FF */
TCNT1 = 0x1FF;
/* 将 TCNT1 读入 i */
i = TCNT1;
...
```

Note: 1. 本代码假定已经包含了合适的头文件。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

汇编代码例程中 TCNT1 的返回值在 r17:r16 寄存器对中。

注意到 16 位寄存器的访问是一个基本操作是非常重要的。在对 16 位寄存器操作时，最好首先屏蔽中断响应，防止在主程序读写 16 位寄存器的两条指令之间发生这样的中断：它也访问同样的寄存器或其他的 16 位寄存器，从而更改了临时寄存器。如果这种情况发生，那么中断返回后临时寄存器中的内容已经改变，造成主程序对 16 位寄存器的读写错误。

下面的例程给出了 TCNT1 寄存器的基本读操作。对 OCR1A/B 或 ICR1 的读操作可以使用相同的方法。

汇编代码例程 (1)

```
TIM16_ReadTCNT1:
  ; 保存全局中断标志
  in  r18,SREG
  ; 禁用中断
  cli
  ; 将 TCNT1 读入 r17:r16
  in  r16,TCNT1L
  in  r17,TCNT1H
  ; 恢复全局中断标志
  out SREG,r18
  ret
```

C 代码例程 (1)

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  /* 保存全局中断标志 */
  sreg = SREG;
  /* 禁用中断*/
  _CLI();
  /* 将 TCNT1 读入 i */
  i = TCNT1;
  /* 恢复全局中断标志 */
  SREG = sreg;
  return i;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

汇编代码例程中 TCNT1 的返回值在 r17:r16 寄存器对中。

下面的例程给出了 TCNT1 寄存器的基本写操作。对 OCR1A/B 或 ICR1 的写操作可以使用相同的方法。

汇编代码例程 (1)

```
TIM16_WriteTCNT1:
  ; 保存全局中断标志
  in  r18,SREG
  ; 禁用中断
  cli
  ; 设置 TCNT1 到 r17:r16
  out TCNT1H,r17
  out TCNT1L,r16
  ; 恢复全局中断标志
  out SREG,r18
  ret
```

C 代码例程 (1)

```
void TIM16_WriteTCNT1( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* 保存全局中断标志 */
  sreg = SREG;
  /* 禁用中断 */
  _CLI();
  /* 设置 TCNT1 到 i */
  TCNT1 = i;
  /* 恢复全局中断标志 */
  SREG = sreg;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

汇编代码例程中 r17:r16 寄存器对保存的是 TCNT1 的写入数据。

临时高字节寄存器的重用

如果对不只一个 16 位寄存器写入数据而且所有寄存器的高字节都相同，则只需写一次高字节。前面讲到的基本操作在这种情况下同样适用。

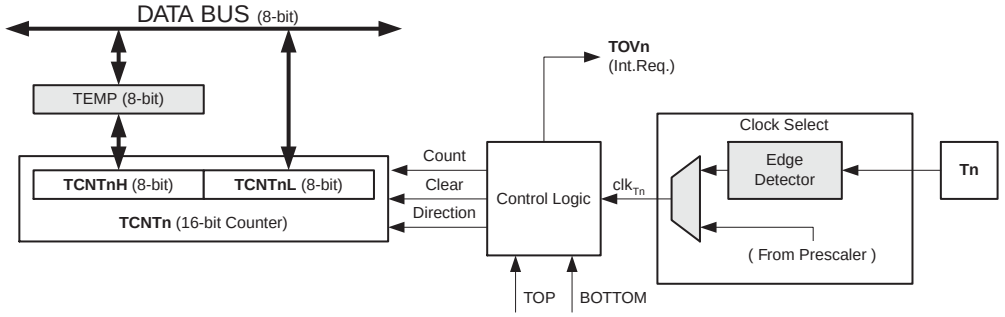
T/C 时钟源

T/C 时钟源可以来自内部，也可来自外部，由位于 T/C 控制寄存器 B(TCCR1B) 的时钟选择位 (CS12:0) 决定。时钟源与预分频器的描述见 P95”T/C0 与 T/C1 的预分频器 ”。

控制单元

16 位 T/C 的主要部分是可编程的 16 位双向计数器单元。 Figure42 给出了计数器与其外围电路方框图。

Figure 42. 计数器单元方框图



信号描述 (内部信号) :

Count	TCNT1 加 1 或减 1
Direction	确定是加操作还是减操作
Clear	TCNT1 清零
clk_{T1}	定时器 / 计数器时钟信号
TOP	表示 TCNT1 计数器到达最大值
BOTTOM	表示 TCNT1 计数器到达最小值 (0)

16 位计数器映射到两个 8 位 I/O 存储器位置:TCNT1H 为高 8 位，TCNT1L 为低 8 位。CPU 只能间接访问 TCNT1H 寄存器。 CPU 访问 TCNT1H 时，实际访问的是临时寄存器 (TEMP)。读取TCNT1L时，临时寄存器的内容更新为TCNT1H的数值；而对TCNT1L执行写操作时，TCNT1H 被临时寄存器的内容所更新。这就使 CPU 可以在一个时钟周期里通过 8 位数据总线完成对 16 位计数器的读、写操作。此外还需要注意计数器在运行时的一些特殊情况。在这些特殊情况下对 TCNT1 写入数据会带来未知的结果。在合适的章节会对这些特殊情况进行具体描述。

根据不同的工作模式，计数器在每一个 clk_{T1} 时钟进行清零、加 1 或减 1 操作。clk_{T1} 由时钟选择位 CS12:0 设定。当 CS12:0= 0 时，计数器停止计数。不过 CPU 对 TCNT1 的读取与 clk_{T1} 是否存在无关。 CPU 写操作比计数器清零和其他操作的优先级都高。

计数器的计数序列取决于寄存器 TCCR1A 和 TCCR1B 中标志位 WGM13:0 的设置。计数器的运行 (计数) 方式与通过 OC1x 输出的波形发生方式有很紧密的关系。计数序列与波形产生的详细描述请参见 P107” 工作模式 ”。

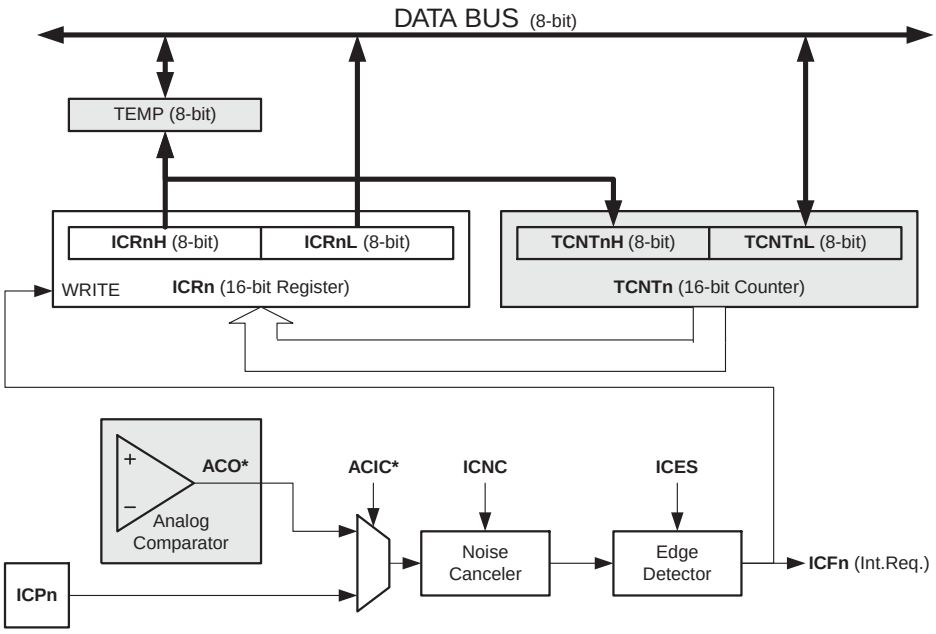
通过 WGM13:0 确定了计数器的工作模式之后，TOV1 的置位方式也就确定了。TOV1 可以用来产生 CPU 中断。

输入捕捉单元

T/C 的输入捕捉单元可用来捕获外部事件，并为其赋予时间标记以说明此时间的发生时刻。外部事件发生的触发信号由引脚 ICP1 输入，也可通过模拟比较器单元来实现。时间标记可用来计算频率、占空比及信号的其它特征，以及为事件创建日志。

输入捕捉单元方框图见 Figure43。图中不直接属于输入捕捉单元的部分用阴影表示。寄存器与位中的小写 “n” 表示定时器 / 计数器编号。

Figure 43. 输入捕捉单元方框图



当引脚 ICP1 上的逻辑电平（事件）发生了变化，或模拟比较器输出 ACO 电平发生了变化，并且这个电平变化为边沿检测器所证实，输入捕捉即被激发：16 位的 TCNT1 数据被复制到输入捕捉寄存器 ICR1，同时输入捕捉标志位 ICF1 置位。如果此时 ICIE1 = 1，输入捕捉标志将产生输入捕捉中断。中断执行时 ICF1 自动清零，或者也可通过软件在其对应的 I/O 位置写入逻辑 "1" 清零。

读取 ICR1 时要先读低字节 ICR1L，然后再读高字节 ICR1H。读低字节时，高字节被复制到高字节临时寄存器 TEMP。 CPU 读取 ICR1H 时将访问 TEMP 寄存器。

对 ICR1 寄存器的写访问只存在于波形产生模式。此时 ICR1 被用作计数器的 TOP 值。写 ICR1 之前首先要设置 WGM13:0 以允许这个操作。对 ICR1 寄存器进行写操作时必须先将高字节写入 ICR1H I/O 位置，然后再将低字节写入 ICR1L。

请参见 P99" 访问 16 位寄存器 " 以了解更多的关于如何访问 16 位寄存器的信息。

输入捕捉触发源

输入捕捉单元的主要触发源是 ICP1。T/C1 还可用模拟比较输出作为输入捕捉单元的触发源。用户必须通过设置模拟比较控制与状态寄存器 ACSR 的模拟比较输入捕捉位 ACIC 来做到这一点。要注意的是，改变触发源有可能造成一次输入捕捉。因此在改变触发源后必须对输入捕捉标志执行一次清零操作以避免出现错误的结果。

CP1 与 ACO 的采样方式与 T1 引脚是相同的 (P94Figure 39), 使用的边沿检测器也一样。但是使能噪声抑制器后，在边沿检测器前会加入额外的逻辑电路并引入 4 个系统时钟周期的延迟。要注意的是，除去使用 ICR1 定义 TOP 的波形产生模式外， T/C 中的噪声抑制器与边沿检测器总是使能的。

输入捕捉也可以通过软件控制引脚 ICP1 的方式来触发。

噪声抑制器

噪声抑制器通过一个简单的数字滤波方案提高系统抗噪性。它对输入触发信号进行 4 次采样。只有当 4 次采样值相等时其输出才会送入边沿检测器。

置位 TCCR1B 的 ICNC1 将使能噪声抑制器。使能噪声抑制器后，在输入发生变化到 ICR1 得到更新之间将会有额外的 4 个系统时钟周期的延时。噪声抑制器使用的是系统时钟，因而不受预分频器的影响。

输入捕捉单元的使用

使用输入捕捉单元的最大问题就是分配足够的处理器资源来处理输入事件。事件的时间间隔是关键。如果处理器在下一次事件出现之前没有读取 ICR1 的数据，ICR1 就会被新值覆盖，从而无法得到正确的捕捉结果。

使用输入捕捉中断时，中断程序应尽可能早的读取 ICR1 寄存器。尽管输入捕捉中断优先级相对较高，但最大中断响应时间与其它正在运行的中断程序所需的时间相关。

在任何输入捕捉工作模式下都不推荐在操作过程中改变 TOP 值。

测量外部信号的占空比时要求每次捕捉后都要改变触发沿。因此读取 ICR1 后必须尽快改变敏感的信号边沿。改变边沿后，ICF1 必须由软件清零 (在对应的 I/O 位置写 "1")。若仅需测量频率，且使用了中断，则不需对 ICF1 进行软件清零。

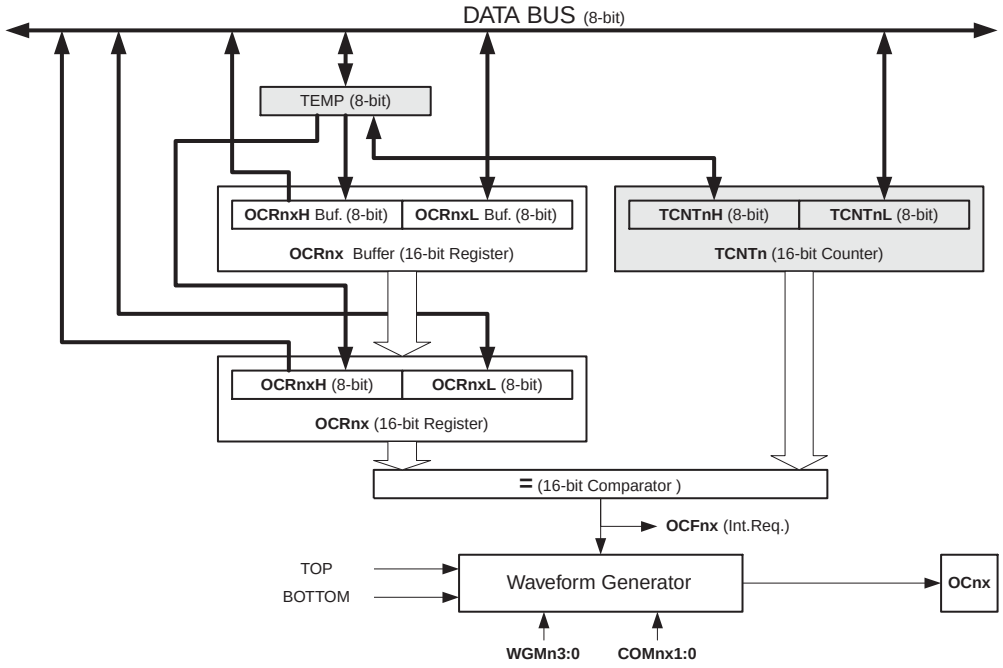
输出比较单元

16 位比较器持续比较 TCNT1 与 OCR1x 的内容，一旦发现它们相等，比较器立即产生一个匹配信号。然后 OCF1x 在下一个定时器时钟置位。如果此时 OCIE1x = 1，OCF1x 置位将引发输出比较中断。中断执行时 OCF1x 标志自动清零，或者通过软件在其相应的 I/O 位置写入逻辑 "1" 也可以清零。根据 WGM13:0 与 COM1x1:0 的不同设置，波形发生器用匹配信号生成不同的波形。波形发生器利用 TOP 和 BOTTOM 信号处理在某些模式下对极值的操作 (P107“ 工作模式 ”)。

输出比较单元 A 的一个特质是定义 T/C 的 TOP 值 (即计数器的分辨率)。此外，TOP 值还用来定义通过波形发生器产生的波形的周期。

Figure44 给出输出比较单元的方框图。寄存器与位上的小写 “n” 表示器件编号 (n = 1 表示 T/C1)，“x” 表示输出比较单元 (A/B)。框图中非输出比较单元部分用阴影表示。

Figure 44. 输出比较单元方框图



当 T/C 工作于 12 种 PWM 模式的任意一种时，OCR1x 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式 (CTC) 双缓冲功能是禁止的。双缓冲可以实现 OCR1x 寄存器对 TOP 或 BOTTOM 的同步更新，防止产生不对称的 PWM 波形，消除毛刺。

访问 OCR1x 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCR1x 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR1x 本身。OCR1x(缓冲或比较) 寄存器的内容只有写操作才能将其改变 (T/C 不会自动将此寄存器更新为 TCNT1 或 ICR1

的内容），所以 OCR1x 不用通过 TEMP 读取。但是象其他 16 位寄存器一样首先读取低字节是一个好习惯。由于比较是连续进行的，因此在写 OCR1x 时必须通过 TEMP 寄存器来实现。首先需要写入的是高字节 OCR1xH。当 CPU 将数据写入高字节的 I/O 地址时，TEMP 寄存器的内容即得到更新。接下来写低字节 OCR1xL。在此同时，位于 TEMP 寄存器的高字节数据被拷贝到 OCR1x 缓冲器，或是 OCR1x 比较寄存器。

请参见 P99” 访问 16 位寄存器 ” 以了解更多的关于如何访问 16 位寄存器的信息。

强制输出比较

工作于非 PWM 模式时，可以通过对强制输出比较位 FOC1x 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCF1x 标志，也不会重载 / 清零定时器，但是 OC1x 引脚将被更新，好象真的发生了比较匹配一样 (COMx1:0 决定 OC1x 是置位、清零，还是交替变化)。

写 TCNT1 操作阻止比较匹配

CPU 对 TCNT1 寄存器的写操作将在下一个定时器时钟周期里阻止比较匹配的发生。这个特性可以用来将 OCR1x 初始化为与 TCNT1 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT1 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变TCNT1就会有风险，不管T/C是否在运行。若写入TCNT1的数值等于OCR1x，比较匹配就被忽略了，造成不正确的波形发生结果。在 PWM 模式下，当 TOP 为可变数值时，不要赋予 TCNT1 和 TOP 相等的数值。否则可能会丢失一次比较匹配，计数器也将计到 0xFFFF。类似地，在计数器进行降序计数时不要对 TCNT1 写入等于 BOTTOM 的数据。

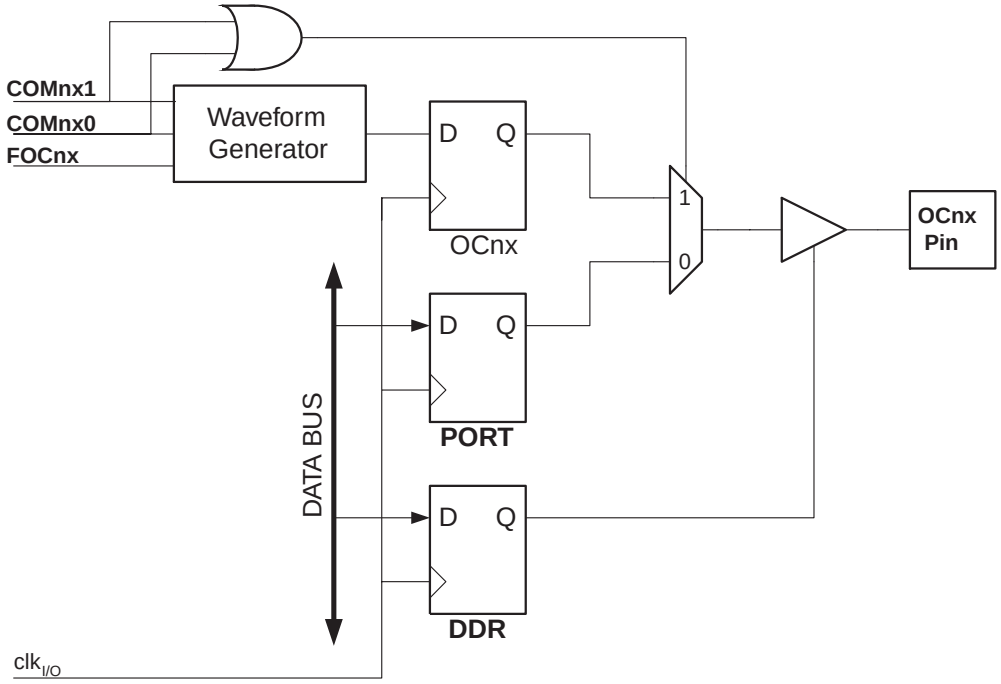
OC1x 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC1x 的方法是在普通模式下利用强制输出比较 FOC1x。即使在改变波形发生模式时 OC1x 寄存器也会一直保持它的数值。

要注意 COM1x1:0 和比较数据都不是双缓冲的。 COM1x1:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM1x1:0 具有双重功能。波形发生器利用 COM1x1:0 来确定下一次比较匹配发生时的输出比较 OC1x 状态； COM1x1:0 还控制 OC1x 引脚输出的来源。Figure45 为受 COM1x1:0 设置影响的逻辑的简化原理图。I/O 寄存器、I/O 位和 I/O 引脚以粗体表示。图中只给出了受 COM1x1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC1x 状态时指的是内部 OC1x 寄存器，而不是 OC1x 引脚的状态。系统复位时 COM1x 寄存器复位为 "0”。

Figure 45. 比较匹配输出单元原理图



只要 COM1x1:0 不全为零，波形发生器的输出比较功能就会重载 OC1x 的通用 I/O 口功能。但是 OC1x 引脚的方向仍旧受控于数据方向寄存器 (DDR)。从 OC1x 引脚输出有效信号之前必须通过数据方向寄存器的 DDR_OC1x 将此引脚设置为输出。一般情况下功能重载与波形发生器的工作模式无关，但也有一些例外，详见 Table55、Table56 与 Table57。

输出比较逻辑的设计允许 OC1x 在输出之前首先进行初始化。要注意某些 COM1x1:0 设置在某些特定的工作模式下是保留的，如 P117“16位定时器/计数器寄存器的说明” 所示。

COM1x1:0 不影响输入捕捉单元。

比较输出模式和波形产生

波形发生器利用 COM1x1:0 的方法在普通模式、 CTC 模式和 PWM 模式下有所区别。对于所有的模式 , 设置 COM1x1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC1x 寄存器。非 PWM 模式的比较输出请参见 P116Table55 ，快速 PWM 的比较输出于 P116Table56 ，相位修正 PWM 的比较输出于 P117Table57 .

改变 COM1x1:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC1x 来立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM13:0) 及比较输出模式 (COM1x1:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM1x1:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COM1x1:0 控制输出是否应该在比较匹配发生时置位、清零 , 或是电平取反 (见 P106“ 比较匹配输出单元 ”)。

具体的时序信息请参考 P115” 定时器 / 计数器时序图 ” 。

普通模式

普通模式 (WGM13:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到最大值后 (MAX = 0xFFFF) 由于数值溢出计数器简单地返回到最小值 0x0000 重新开始。在TCNT1为零的同一个定时器时钟里T/C溢出标志TOV1置位。此时TOV1有点象第17位 , 只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV1，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的 , 用户可以随时写入新的计数器数值。

在普通模式下输入捕捉单元很容易使用。要注意的是外部事件的最大时间间隔不能超过计数器的分辨率。如果事件间隔太长 , 必须使用定时器溢出中断或预分频器来扩展输入捕捉单元的分辨率。

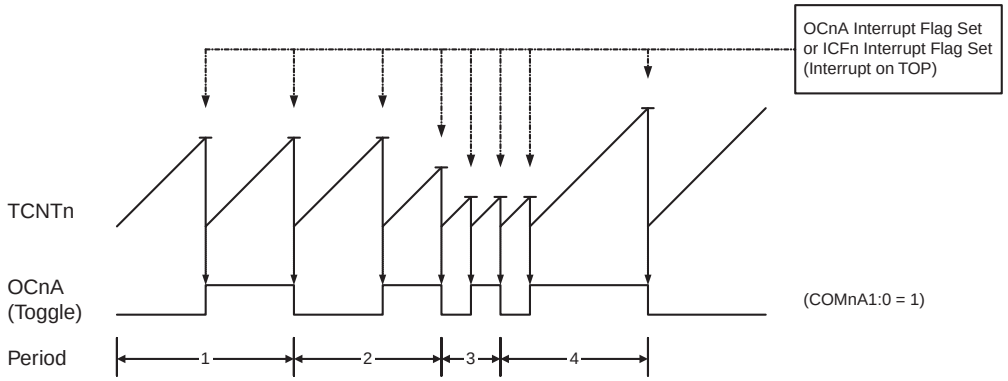
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形 , 因为这会占用太多的 CPU 时间。

CTC(比较匹配时清零定时器) 模式

在 CTC 模式 (WGM13:0 = 4 或 12) 里 OCR1A 或 ICR1 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT1 等于 OCR1A(WGM13:0 = 4) 或等于 ICR1 (WGM13:0 = 12) 时计数器清零。 OCR1A 或 ICR1 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为 Figure46。计数器数值TCNT1一直累加到TCNT1与OCR1A 或ICR1匹配，然后 TCNT1 清零。

Figure 46. CTC 模式的时序图



利用 OCF1A 或 ICF1 标志可以在计数器数值达到 TOP 时产生中断。在中断服务程序里可以更新 TOP 的数值。由于 CTC 模式没有双缓冲功能，在计数器以无预分频器或很低的预分频系数工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入的 OCR1A 或 ICR1 的数值小于当前 TCNT1 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFFFF，然后再从 0x0000 开始计数到 OCR1A 或ICR1。在许多情况下，这种情况并非我们所希望的，解决的办法是在快速PWM模式下使用 OCR1A 来定义 TOP (WGM13:0 = 15) 值，因为 OCR1A 有双缓冲。

为了在CTC模式下得到波形输出，可以设置OC0A在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM1A1:0 = 1 来完成。在期望获得 OC1A 输出之前，首先要将其端口设置为输出(DDR_OC1A = 1)。波形发生器能够产生的最大频率为 $f_{OC2} = f_{clk_I/O}/2$ (OCR1A = 0x0000)。频率由如下公式确定：

$$f_{OCnA} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

变量 N 代表预分频因子 (1、 8、 64、 256 或 1024)。

在普通模式下，TOV1 标志的置位发生在计数器从 MAX 变为 0x0000 的定时器时钟周期。

快速 PWM 模式

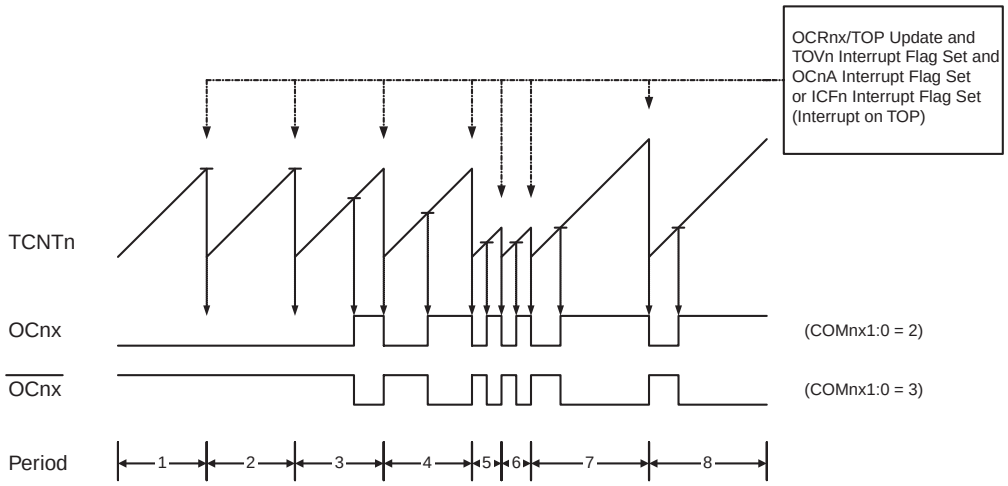
快速 PWM 模式 (WGM13:0 = 5、 6、7、 14 或 15) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 MAX ,然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式 ,输出比较引脚 OC1x 在 TCNT1 与 OCR1x 匹配时置位，在 TOP 时清零；对于反向比较输出模式， OCR1x 的动作正好相反。由于使用了单边斜坡模式 ,快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节 ,整流和 DAC 应用。高频可以减小外部元器件 (电感 ,电容) 的物理尺寸 ,从而降低系统成本。

工作于快速 PWM 模式时 ,PWM 分辨率可固定为 8、9 或 10 位 ,也可由 ICR1 或 OCR1A 定义。最小分辨率为 2 比特 (ICR1 或 OCR1A 设为 0x0003)，最大分辨率为 16 位 (ICR1 或 OCR1A 设为 MAX)。 PWM 分辨率位数可用下式计算：

$$R_{FPWM} = \frac{\log(TOP+1)}{\log(2)}$$

工作于快速 PWM 模式时 ,计数器的数值一直累加到固定数值 0x00FF、0x01FF、0x03FF (WGM13:0 = 5、6 或 7)、ICR1 (WGM13:0 = 14) 或 OCR1A (WGM13:0 = 15) ,然后在紧接的时钟周期清零。具体的时序图为 Figure47。图中给出了当使用 OCR1A 或 ICR1 来定义 TOP 值时的快速 PWM 模式。图中柱状的 TCNT1 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。TCNT1 斜坡上的短水平线表示 OCR1x 和 TCNT1 的匹配比较。比较匹配后 OC1x 中断标志置位。

Figure 47. 快速 PWM 模式时序图



计时器数值达到 TOP 时 T/C 溢出标志 TOV1 置位。另外若 TOP 值是由 OCR1A 或 ICR1 定义的 ,则 OC1A 或 ICF1 标志将与 TOV1 在同一个时钟周期置位。如果中断使能，可以在中断服务程序里更新 TOP 以及比较数据。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNT1与OCR1x不会出现比较匹配。使用固定的 TOP 值时，向任意 OCR1x 寄存器写入数据时未使用的位将屏蔽为 "0”。

定义 TOP 值时更新 ICR1 与 OCR1A 的步骤时不同的。ICR1 寄存器不是双缓冲寄存器。这意味着当计数器以无预分频器或很低的预分频工作的时候，给 ICR1 赋予一个小的数值时存在着新写入的 ICR1 数值比 TCNT1 当前值小的危险。结果是计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFFFF，然后再从 0x0000 开始计数 ,直到比较匹配出现。而 OCR1A 寄存器则是双缓冲寄存器。这一特性决定 OCR1A 可以随时写入。写入的数据被放入 OCR1A 缓冲寄存器。在 TCNT1 与 TOP 匹配后的下一个时钟周期，OCR1A 比较寄存器的内容被缓冲寄存器的数据所更新。在同一个时钟周期 TCNT1 被清零，而 TOV1 标志被设置。

使用固定 TOP 值时最好使用 ICR1 寄存器定义 TOP。这样 OCR1A 就可以用于在 OC1A 输出 PWM 波。但是，如果 PWM 基频不断变化 (通过改变 TOP 值)， OCR1A 的双缓冲特性使其更适合于这个应用。

工作于快速 PWM 模式时 , 比较单元可以在 OC1x 引脚上输出 PWM 波形。设置 COM1x1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形 (参见 P116Table)。此外，要真正从物理引脚上输出信号还必须将 OC1x 的数据方向 DDR_OC1x 设置为输出。产生 PWM 波形的机理是 OC1x 寄存器在 OCR1x 与 TCNT1 匹配时置位 (或清零)，以及在计数器清零 (从 TOP 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot (1 + TOP)}$$

变量 N 代表分频因子 (1、8、64、 256 或 1024)。

OCR1x 寄存器为极限值时说明了快速 PWM 模式的生成 PWM 波形输出一些特殊情况。若 OCR1x 等于 BOTTOM(0x0000)，输出为出现在第 TOP+1 个定时器时钟周期的窄脉冲；OCR1x 为 TOP 时，根据 COM1x1:0 的设定，输出恒为高电平或低电平。

通过设定 OC1A 在比较匹配时进行逻辑电平取反 (COM1A1:0 = 1)，可以得到占空比为 50% 的周期信号。这只适用于 OCR1A 用来定义 TOP 值的情况 (WGM13:0 = 15)。OCR1A 为 0(0x0000) 时信号有最高频率 $f_{oc2} = f_{clk_I/O}/2$。这个特性类似于 CTC 模式下的 OC1A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

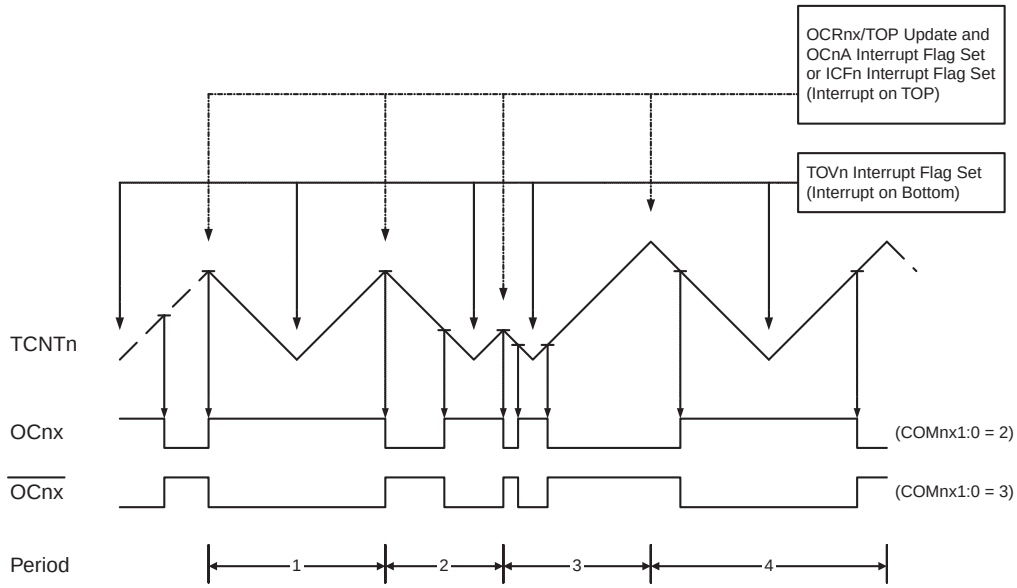
相位修正 PWM 模式 (WGM13:0 = 1、2、3、10 或 11) 为用户提供了一个获得高精度的、相位准确的 PWM 波形的方法。与相位和频率修正模式类似，此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。在一般的比较输出模式下，当计时器往 TOP 计数时若 TCNT1 与 OCR1x 匹配，OC1x 将清零为低电平；而在计时器往 BOTTOM 计数时若 TCNT1 与 OCR1x 匹配， OC1x 将置位为高电平。工作于反向比较输出时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

相位修正 PWM 模式的 PWM 分辨率固定为 8、 9 或 10 位，或由 ICR1 或 OCR1A 定义。最小分辨率为 2 比特 (ICR1 或 OCR1A 设为 0x0003)，最大分辨率为 16 位 (ICR1 或 OCR1A 设为 MAX)。 PWM 分辨率位数可用下式计算：

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于相位修正 PWM 模式时，计数器的数值一直累加到固定值 0x00FF、 0x01FF、0x03FF (WGM13:0 = 1、2 或 3)、ICR1 (WGM13:0 = 10) 或 OCR1A (WGM13:0 = 11)，然后改变计数方向。在一个定时器时钟周期里 TCNT1 值等于 TOP 值。具体的时序图为 Figure48。图中给出了当使用OCR1A或ICR1来定义TOP值时的相位修正PWM模式。图中柱状的 TCNT1 表示这是双边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。TCNT1 斜坡上的短水平线表示 OCR1x 和 TCNT1 的匹配比较。比较匹配后 OC1x 中断标志置位。

Figure 48. 相位修正 PWM 模式的时序图



计时器数值达到 BOTTOM 时 T/C 溢出标志 TOV1 置位。若 TOP 由 OCR1A 或 ICR1 定义，在 OCR1x 寄存器通过双缓冲方式得到更新的同一个时钟周期里 OC1A 或 ICF1 标志置位。标志置位后即可产生中断。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNT1与OCR1x不会出现比较匹配。使用固定的 TOP 值时，向任意 OCR1x 寄存器写入数据时未使用的位将屏蔽为 "0”。在 Figure48 给出的第三个周期中，在 T/C 运行于相位修正模式时改变 TOP 值导致了不对称输出。其原因在于 OCR1x 寄存器的更新时间。由于 OCR1x 的更新时刻为定时器 / 计数器达到 TOP 之时，因此 PWM 的循环周期起始于此，也终止于此。就是说，下降斜坡的长度取决于上一个 TOP 值，而上升斜坡的长度取决于新的 TOP 值。若这两个值不同，一个周期内两个斜坡长度不同，输出也就不对称了。

若要在 T/C 运行时改变 TOP 值，最好用相位与频率修正模式代替相位修正模式。若 TOP 保持不变，那么这两种工作模式实际上没有区别。

工作于相位修正 PWM 模式时，比较单元可以在 OC1x 引脚输出 PWM 波形。设置 COM1x1:0 为 2 可以产生普通的 PWM，设置 COM1x1:0 为 3 可以产生反向 PWM (参见 P117Table)。要真正从物理引脚上输出信号还必须将 OC1x 的数据方向 DDR_OC1x 设置为输出。OCR1x 和 TCNT1 比较匹配发生时 OC1x 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由如下公式获得：

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

变量 N 表示预分频因子 (1、8、64、256 或 1024)。

OCR1x 寄存器处于极值时表明了相位修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR1x 等于 BOTTOM，输出一直保持为低电平；若 OCR1x 等于 TOP，输出则保持为高电平。反向 PWM 模式正好相反。如果 OCR1A 用来定义 TOP 值 (WGM13:0 = 11) 且 COM1A1:0 = 1， OC1A 输出占空比为 50% 的周期信号。

相位与频率修正 PWM 模式

相位与频率修正 PWM 模式 (WGM13:0 = 8 或 9) - 以下简称相频修正 PWM 模式 - 可以产生高精度的、相位与频率都准确的 PWM 波形。与相位修正模式类似，相频修正 PWM 模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒计数到 BOTTOM。在一般的比较输出模式下，当计时器往TOP计数时若TCNT1与OCR1x匹配，OC1x将清零为低电平；而在计时器往BOTTOM计数时TCNT1与OCR1x匹配，OC1x将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

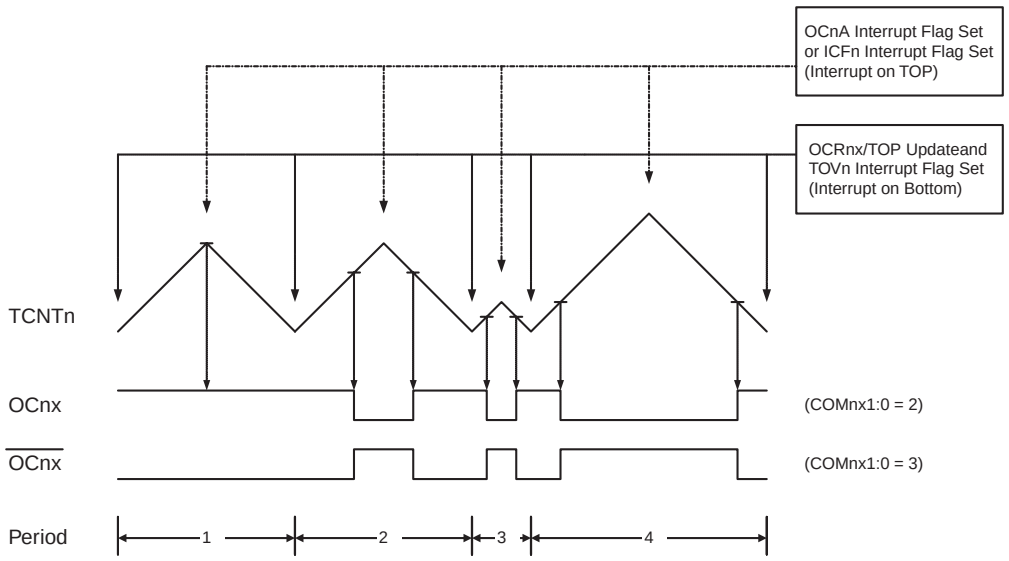
相频修正修正 PWM 模式与相位修正 PWM 模式的主要区别在于 OCR1x 寄存器的更新时间，详见 Figure48 与 Figure49。

相频修正修正 PWM 模式的 PWM 分辨率可由 ICR1 或 OCR1A 定义。最小分辨率为 2 比特 (ICR1 或 OCR1A 设为 0x0003)，最大分辨率为 16 位 (ICR1 或 OCR1A 设为 MAX)。PWM 分辨率位数可用下式计算：

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于相频修正PWM模式时，计数器的数值一直累加到ICR1 (WGM13:0 = 8)或OCR1A (WGM13:0 = 9)，然后改变计数方向。在一个定时器时钟里TCNT1值等于TOP值。具体的时序图为 Figure49。图中给出了当使用 OCR1A 或 ICR1 来定义 TOP 值时的相频修正PWM模式。图中柱状的TCNT1表示这是双边斜坡操作。方框图同时包含了普通的PWM输出以及反向 PWM 输出。TCNT1 斜坡上的短水平线表示 OCR1x 和 TCNT1 的匹配比较。比较匹配发生时， OC1x 中断标志将被置位。

Figure 49. 相位与频率修正 PWM 模式的时序图



在 OCR1x 寄存器通过双缓冲方式得到更新的同一个时钟周期里 T/C 溢出标志 TOV1 置位。若 TOP 由 OCR1A 或 ICR1 定义，则当 TCNT1 达到 TOP 值时 OC1A 或 CF1 置位。这些中断标志位可用来在每次计数器达到 TOP 或 BOTTOM 时产生中断。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNT1与OCR1x不会产生比较匹配。

如 Figure49 所示，与相位修正模式形成对照的是，相频修正 PWM 模式生成的输出在所有的周期中均为对称信号。这是由于 OCR1x 在 BOTTOM 得到更新，上升与下降斜坡长度始终相等。因此输出脉冲是对称的，确保了频率的正确。

使用固定 TOP 值时最好使用 ICR1 寄存器定义 TOP。这样 OCR1A 就可以用于在 OC1A 输出 PWM 波。但是，如果 PWM 基频不断变化 (通过改变 TOP 值)， OCR1A 的双缓冲特性使其更适合于这个应用。

工作于相频修正 PWM 模式时，比较单元可以在 OC1x 引脚上输出 PWM 波形。设置 COM1x1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形。(参见 P117Table)。要想真正输出信号还必须将 OC1x 的数据方向设置为输出。产生 PWM 波形的机理是 OC1x 寄存器在 OCR1x 与升序记数的 TCNT1 匹配时置位 (或清零) ,与降序记数的 TCNT1 匹配时清零 (或置位)。输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

变量 N 代表分频因子 (1、8、64、 256 或 1024)。

OCR1x 寄存器处于极值时说明了相频修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR1x 等于 BOTTOM，输出一直保持为低电平；若 OCR1x 等于 TOP，则输出保持为高电平。反向 PWM 模式则正好相反。如果 OCR1A 用来定义 TOP 值 (WGM13:0 = 9) 且 COM1A1:0 = 1， OC1A 输出占空比为 50% 的周期信号。

定时器 / 计数器时序图

定时器 / 计数器为同步电路，因而时钟 clk_{T1} 表示为时钟使能信号。图中说明了何时设置中断标志及何时使用 OCR1x 缓冲器中的数据更新 OCR1x 寄存器（工作于双缓冲器模式时）。 Figure50 给出了置位 OCF1x 的时序图。

Figure 50. T/C 时序图 ， OCF1x 置位，无预分频器

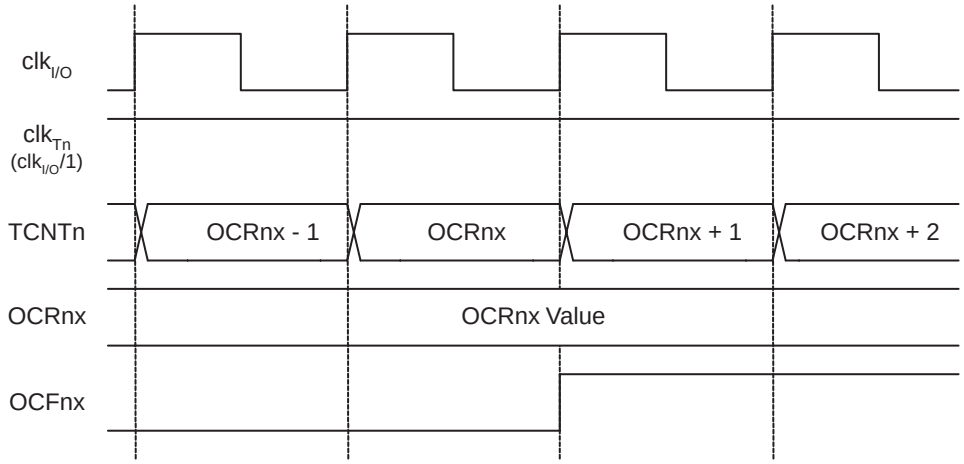


Figure51 给出相同的时钟数据，但预分频使能。

Figure 51. T/C 时序图，置位 OCF1x，预分频器为 $f_{clk_I/O}/8$

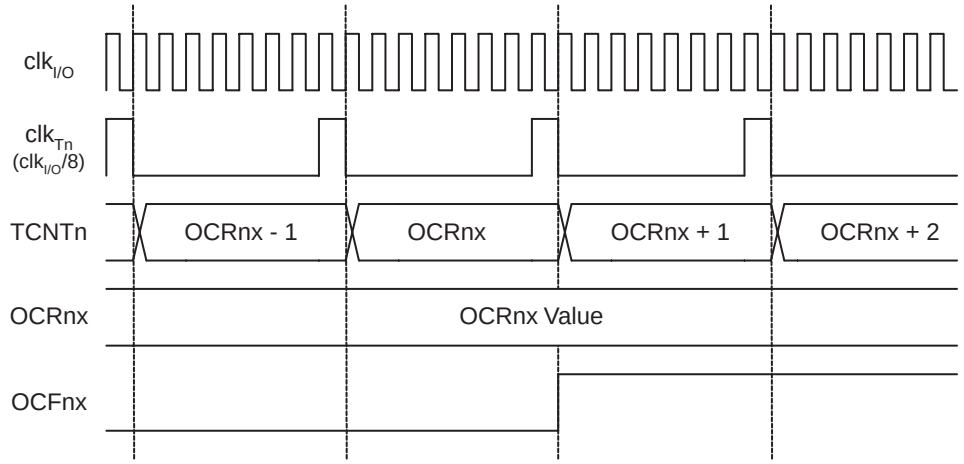


Figure52 给出工作在不同模式下接近 TOP 值时的计数序列。工作于相频修正 PWM 模式时， OCR1x 寄存器在 BOTTOM 被更新。时序图相同，但 TOP 需要用 BOTTOM 代替，BOTTOM+1 代替 TOP-1 ,等等。同样的命名规则也适用于那些在 BOTTOM 置位 TOV1 标志的工作模式。

Figure 52. T/C 时序图，无预分频器

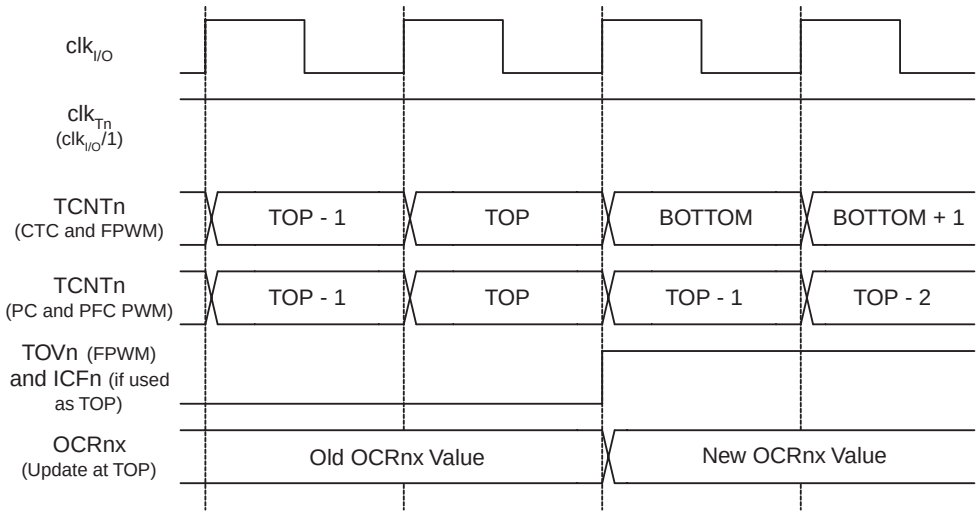
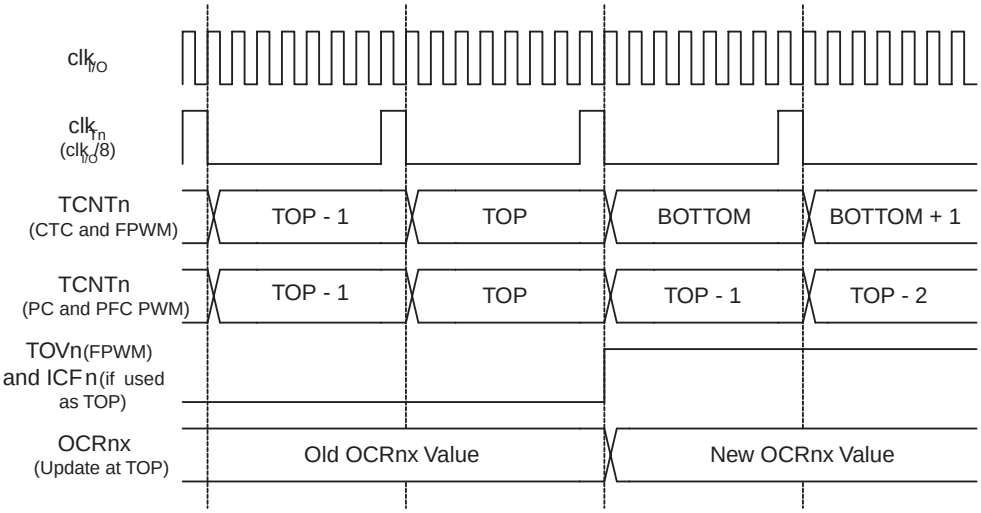


Figure53 给出相同的时钟数据，但预分频使能。

Figure 53. T/C 时序图，预分频器为 $f_{clk_I/O}/8$



16 位定时器 / 计数器寄存器的说明

T/C1 控制寄存器 A—TCCR1A

位	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
读 / 写	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7:6 – COM1A1:0: 通道 A 的比较输出模式**
- **位 5:4 – COM1B1:0: 通道 B 的比较输出模式**

COM1A1:0与COM1B1:0分别控制OC1A 与OC1B的状态。如果COM1A1:0(COM1B1:0)的一位或两位被写入"1”,OC1A(OC1B)输出功能将取代I/O端口功能。此时OC1A(OC1B)相应的输出引脚数据方向控制必须置位以使能输出驱动器。

OC1A(OC1B) 与物理引脚相连时 , COM1x1:0 的功能由 WGM13:0 的设置决定。Table55 给出当 WGM13:0 设置为普通模式与 CTC 模式 (非 PWM) 时 COM1x1:0 的功能定义。

Table 55. 比较输出模式，非 PWM

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作，非 OC1A/OC1B 功能
0	1	比较匹配时 OC1A/OC1B 电平取反
1	0	比较匹配时清零 OC1A/OC1B(输出低电平)
1	1	比较匹配时置位 OC1A/OC1B (输出高电平)

Table56 给出 WGM13:0 设置为快速 PWM 模式时 COM1x1:0 的功能定义。

Table 56. 比较输出模式，快速 PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作，非 OC1A/OC1B 功能
0	1	WGM13:0 = 15: 比较匹配时 OC1A 取反 , OC1B 不占用物理引脚。 WGM1 为其它值时为普通端口操作，非 OC1A/OC1B 功能
1	0	比较匹配时清零 OC1A/OC1B，OC1A/OC1B 在 TOP 时置位
1	1	比较匹配时置位 OC1A/OC1B， OC1A/OC1B 在 TOP 时清零

Note: 1. 当 OCR1A/OCR1B 等于 TOP 且 COM1A1/COM1B1 置位时，比较匹配被忽略，但 OC1A/OC1B 的置位 / 清零操作有效。详见 P109“ 快速 PWM 模式 ” 。

Table57 给出当WGM13:0设置为相位修正PWM模式或相频修正PWM模式时COM1x1:0的功能定义。

Table 57. 比较输出模式，相位修正及相频修正 PWM 模式 (1)

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作，非 OC1A/OC1B 功能
0	1	WGM13:0 = 9 或 14: 比较匹配时 OC1A 取反，OC1B 不占用物理引脚。WGM1 为其它值时为普通端口操作，非 OC1A/OC1B 功能
1	0	升序记数时比较匹配将清零 OC1A/OC1B，降序记数时比较匹配将置位 OC1A/OC1B
1	1	升序记数时比较匹配将置位 OC1A/OC1B，降序记数时比较匹配将清零 OC1A/OC1B

Note: 1. OCR1A/OCR1B 等于 TOP 且 COM1A1/COM1B1 置位是一个特殊情况。详细信息请参见 P111“ 相位修正 PWM 模式 ” 。

- **位 1:0 – WGM11:0: 波形发生模式**

这两位与位于 TCCR1B 寄存器的 WGM13:2 相结合，用于控制计数器的计数序列——计数器计数的上限值和确定波形发生器的工作模式 (见 Table58)。T/C 支持的工作模式有 : 普通模式 (计数器)，比较匹配时清零定时器 (CTC) 模式，及三种脉宽调制 (PWM) 模式 (P107“ 工作模式 ”)。

Table 58. 波形产生模式的位描述 (1)

模式	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	定时器 / 计数器工作模式	计数上限值 TOP	OCR1x 更新时刻	TOV1 标志设置
0	0	0	0	0	普通模式	0xFFFF	立即更新	MAX
1	0	0	0	1	8 位相位修正 PWM	0x00FF	TOP	BOTTOM
2	0	0	1	0	9 位相位修正 PWM	0x01FF	TOP	BOTTOM
3	0	0	1	1	10 位相位修正 PWM	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	立即更新	MAX
5	0	1	0	1	8 位快速 PWM	0x00FF	TOP	TOP
6	0	1	1	0	9 位快速 PWM	0x01FF	TOP	TOP
7	0	1	1	1	10 位快速 PWM	0x03FF	TOP	TOP
8	1	0	0	0	相位与频率修正 PWM	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	相位与频率修正 PWM	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	相位修正 PWM	ICR1	TOP	BOTTOM
11	1	0	1	1	相位修正 PWM	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	立即更新	MAX
13	1	1	0	1	保留	–	–	–
14	1	1	1	0	快速 PWM	ICR1	TOP	TOP
15	1	1	1	1	快速 PWM	OCR1A	TOP	TOP

Note: 1. CTC1 和 PWM11:0 的定义已经不再使用了，要使用 WGM12:0。但是两个版本的功能和位置是兼容的。

T/C1 控制寄存器 B—TCCR1B

位	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – ICNC1: 输入捕捉噪声抑制器**

- 置位 ICNC1 将使能输入捕捉噪声抑制功能。此时外部引脚 ICP1 的输入被滤波。其作用是从 ICP1 引脚连续进行 4 次采样。如果 4 个采样值都相等，那么信号送入边沿检测器。因此使能该功能使得输入捕捉被延迟了 4 个时钟周期。

- **位 6 – ICES1: 输入捕捉触发沿选择**

该位选择使用 ICP1 上的哪个边沿触发捕获事件。 ICES 为 "0” 选择的是下降沿触发输入捕捉； ICES1 为 "1” 选择的是逻辑电平的上升沿触发输入捕捉。

按照 ICES1 的设置捕获到一个事件后，计数器的数值被复制到 ICR1 寄存器。捕获事件还会置为 ICF1。如果此时中断使能，输入捕捉事件即被触发。

当 ICR1 用作 TOP 值 (见 TCCR1A 与 TCCR1B 寄存器中 WGM13:0 位的描述) 时，ICP1 与输入捕捉功能脱开，从而输入捕捉功能被禁用。

- **位 5 – 保留位**

该位保留。为保证与将来器件的兼容性，写 TCCR1B 时，该位必须写入 "0”。

- **位 4:3 – WGM13:2: 波形发生模式**

见 TCCR1A 寄存器中的描述。

- **位 2:0 – CS12:0: 时钟选择**

这 3 位用于选择 T/C 的时钟源，见 Figure50 与 Figure51。

Table 59. 时钟选择位描述

CS12	CS11	CS10	说明
0	0	0	无时钟源 (T/C 停止)
0	0	1	$clk_{I/O}$/1 (无预分频)
0	1	0	$clk_{I/O}$/8 (来自预分频器)
0	1	1	$clk_{I/O}$/64 (来自预分频器)
1	0	0	$clk_{I/O}$/256 (来自预分频器)
1	0	1	$clk_{I/O}$/1024 (来自预分频器)
1	1	0	外部 T1 引脚，下降沿驱动
1	1	1	外部 T1 引脚，上升沿驱动

选择使用外部时钟源后，即使 T1 引脚被定义为输出，其 1 引脚上的逻辑信号电平变化仍然会驱动 T/C1 计数，这个特性允许用户通过软件来控制计数。

T/C1 控制寄存器 C—TCCR1C

位	7	6	5	4	3	2	1	0	
	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
读 / 写	R/W	R/W	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – FOC1A: 强制输出比较 A**

- **位 6 – FOC1B: 强制输出比较 B**

FOC1A/FOC1B位只在WGM13:0位被设置为非PWM模式时才有效。为了保证同以后的器件兼容，在 PWM 模式下写 TCCR1A 寄存器时，该寄存器必须清零。对 FOC1A/FOC1B 写 "1" 将强制波形发生器产生一次成功的比较匹配，并使波形发生器依据 COM1x1:0 的设置而改变 OC1A/OC1B 的输出状态。FOC1A/FOC1B 的作用如同一个选通信号，COM1x1:0 的设置才是最终确定比较匹配结果的因素。

FOC1A/FOC1B 选通信号不会产生任何中断请求，也不会对计数器清零，象使用 OCR1A 为 TOP 值的 CTC 工作模式那样。

FOC1A/FOC1B 的读返回值总为零。

T/C1—TCNT1H 与 TCNT1L

位	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

TCNT1H与TCNT1L组成了T/C1的数据寄存器TCNT1。通过它们可以直接对定时器/计数器单元的 16 位计数器进行读写访问。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。TEMP 是所有的 16 位寄存器共用的，详见 P99“访问 16 位寄存器 ”。

在计数器运行期间修改TCNT1的内容有可能丢失一次TCNT1与OCR1x的比较匹配操作。

写 TCNT1 寄存器将在下一个定时器周期阻塞比较匹配。

输出比较寄存器 1A—OCR1AH 与 OCR1AL

位	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 1B—OCR1BH 与 OCR1BL

位	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

该寄存器中的 16 位数据与 TCNT1 寄存器中的计数值进行连续的比较，一旦数据匹配，将产生一个输出比较中断，或改变 OC1x 的输出逻辑电平。

输出比较寄存器长度为 16 位。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。 TEMP 是所有的 16 位寄存器共用的，详见 P99“ 访问 16 位寄存器 ”。

输入捕捉寄存器 1—ICR1H 与 ICR1L

位	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

当外部引脚ICP1(或T/C1的模拟比较器)有输入捕捉触发信号产生时,计数器TCNT1中的值写入 ICR1 中。 ICR1 的设定值可作为计数器的 TOP 值。

输入捕捉寄存器长度为 16 位。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。 TEMP 是所有的 16 位寄存器共用的，详见 P99“ 访问 16 位寄存器 ”。

T/C1 中断屏蔽寄存器—TIMSK1

位	7	6	5	4	3	2	1	0	
	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
读 / 写	R	R	R/W	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7, 6 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 0。

- **位 5 – ICIE1: T/C1 输入捕捉中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时， T/C1 的输入捕捉中断使能。 T一旦 IFR1 的 ICF1 置位 ,CPU 即开始执行 T/C1 输入捕捉中断服务程序 (见 P48 “ 中断 ”)。

- **位 4, 3 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 0。

- **位 2 – OCIE1B: T/C1 输出比较 B 匹配中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时，使能 T/C1 的输出比较 B 匹配中断使能。一旦 TIFR1 上的 OCF1B 置位， CPU 即开始执行 T/C1 输出比较 B 匹配中断服务程序 (见 P48 “ 中断 ”)。

- **位 1 – OCIE1A: T/C1 输出比较 A 匹配中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时， T/C1 的输出比较 A 匹配中断使能。一旦 TIFR1 上的 OCF1A 置位， CPU 即开始执行 T/C1 输出比较 A 匹配中断服务程序 (见 P48 “ 中断 ”)。

- **位 0 – TOIE1: T/C1 溢出中断使能**

当 TOIE1 与状态寄存器的 I 位同时被设为 ”1” 时， T/C1 的溢出中断使能。一旦 TIFR1 的 TOV1 置位， CPU 即开始执行 T/C1 溢出中断服务程序 (见 P44“ 看门狗定时器 ”)。

T/C1 中断标志寄存器—TIFR1

位	7	6	5	4	3	2	1	0	
	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
读 / 写	R	R	R/W	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7, 6 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 0。

- **位 5 – ICF1: T/C1 输入捕捉标志位**

外部引脚 ICP1 出现捕捉事件时 ICF1 置位。此外，当 ICR1 作为计数器的 TOP 值时，一旦计数器值达到 TOP， ICF1 也置位。

执行输入捕捉中断服务程序时 ICF1 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

- **位 4, 3 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 0。

- **位 2 – OCF1B: T/C1 输出比较 B 匹配标志位**

当 TCNT1 与 OCR1B 匹配成功时，该位被设为 "1”。

强制输出比较 (FOC1B) 不会置位 OCF1B。

执行强制输出比较匹配 B 中断服务程序时 OCF1B 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

- **位 1 – OCF1A: T/C1 输出比较 A 匹配标志位**

当 TCNT1 与 OCR1A 匹配成功时，该位被设为 "1”。

强制输出比较 (FOC1A) 不会置位 OCF1A。

执行强制输出比较匹配 A 中断服务程序时 OCF1A 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

- **位 0 – TOV1: T/C1 溢出标志**

该位的设置与 T/C1 的工作方式有关。工作于普通模式和 CTC 模式时 ,T/C1 溢出时 TOV1 置位。对工作在其它模式下的 TOV1 标志位置位，见 P118Table58 。

执行溢出中断服务程序时 OCF1A 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

8 位具有 PWM 与异步操作功能的定时器 / 计数器 2

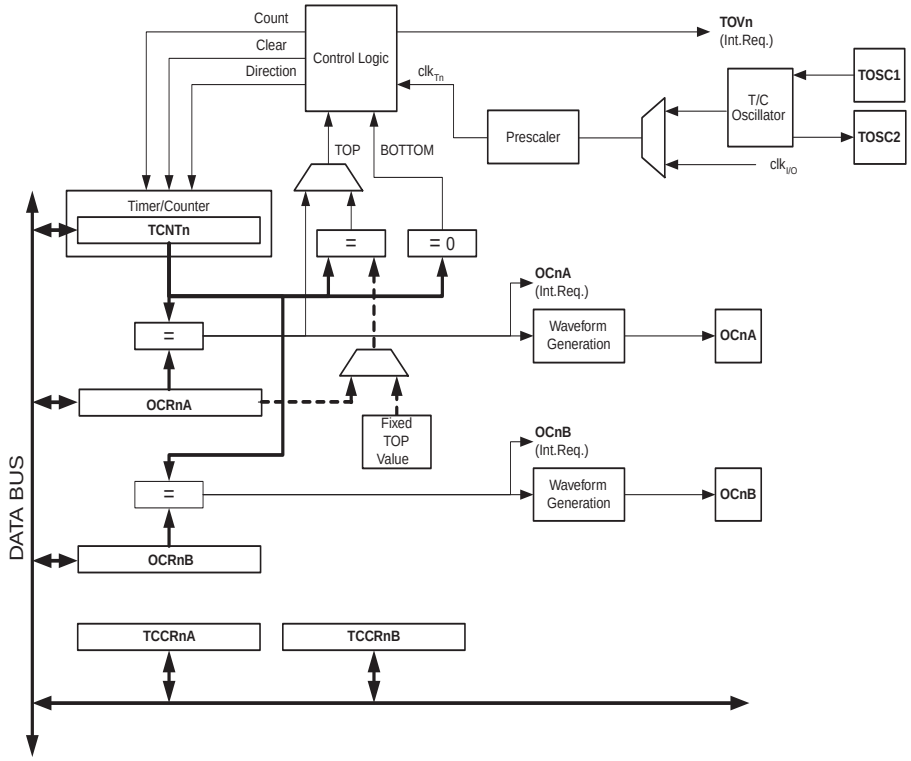
T/C2 是一个通用单通道 8 位定时 / 计数器，其主要特点如下：

- **单通道计数器**
- **比较匹配时清零定时器 (自动重载)**
- **无干扰脉冲 , 相位正确的脉宽调制器 (PWM)**
- **频率发生器**
- **10 位时钟预分频器**
- **溢出与比较匹配中断源 (TOV2 与 OCF2A)**
- **允许使用外部的 32 kHz 手表晶振作为独立的 I/O 时钟源**

综述

Figure54 为 8 位 T/C 的方框图。实际的引脚图请参见 P2”ATmega48/88/168 引脚排列 ”。CPU 可访问的 I/O 寄存器 , 包括 I/O 位和 I/O 引脚以粗体表示。器件具体 I/O 寄存器与位定义见 P134”8 位 T/C 寄存器说明 ”。

Figure 54. 8 位 T/C 方框图



寄存器

定时器 / 计数器 TCNT2、输出比较寄存器 OCR2A 为 8 位寄存器。中断请求 (图中简写为 Int.Req.)。信号在定时器中断标志寄存器 TIFR2 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK2 单独进行屏蔽。图中未给出 TIFR2 与 TIMSK2。

T/C2 的时钟可以为通过预分频器的内部时钟或通过由 TOSC1/2 引脚接入的异步时钟 , 详见本节后续部分。异步操作由异步状态寄存器 ASSR 控制。时钟选择逻辑模块控制引起 T/C 计数值增加(或减少)的时钟源。没有选择时钟源时 T/C 处于停止状态。时钟选择逻辑模块的输出称为 clk_{T2}。

双缓冲的输出比较寄存器 OCR2A 与 OCR2B 持续地与 TCNT2 的数值进行比较。波形发生器利用比较结果产生 PWM 波形或在比较输出引脚 OC2A 与 OCR2B 输出可变频率的信号。参见 P126“输出比较单元 ”。比较匹配结果还会置位比较匹配标志 OCF2A 或 OCF2B，用来产生输出比较中断请求。

定义

本文的许多寄存器及其各个位以通用的格式表示。小写的 “n” 表示定时器 / 计数器的序号，在此即为 2。但是在写程序时要使用精确的格式 (例如使用 TCNT2 来访问 T/C2 计数器值)。

Table 60 的定义适用于本节：

Table 60. 说明

BOTTOM	计数器计到 0x00 时即达到 BOTTOM
MAX	计数器计到 0xFF (十进制的 255) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。TOP 值可以为固定值 0xFF (MAX)，或是存储于寄存器 OCR2A 里的数值，具体由工作模式确定

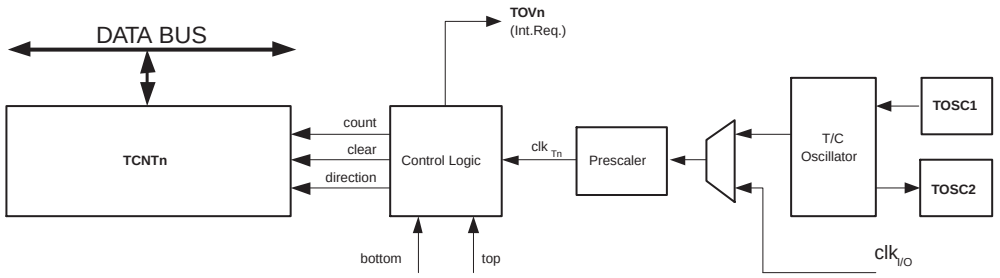
T/C 的时钟源

T/C 可以由内部同步时钟或外部异步时钟驱动。 clk_{T2} 的缺省设置为 MCU 时钟 $clk_{I/O}$。当 ASSR 寄存器的 AS2 置位时，时钟源来自于 TOSC1 和 TOSC2 连接的振荡器。详细的异步操作可以参考 P139” 异步状态寄存器 – ASSR” ；详细的时钟源与预分频器的内容请参考 P142” 定时器 / 计数器预分频器 ” 。

计数器单元

8 位 T/C 的主要部分为可编程的双向计数单元。Figure55 即为计数器和它周边电路的方框图。

Figure 55. 计数器单元方框图



信号说明 (内部信号) :

count 使 TCNT2 加 1 或减 1

direction 选择加操作或减操作

clear 清零 TCNT2 (将所有的位清零)

clk_{T2} T/C 的时钟

top 表示 TCNT2 已经达到了最大值

bottom 表示 TCNT2 已经达到了最小值 (0)

根据不同的工作模式，计数器针对每一个 clk_{T2} 实现清零、加一或减一操作。clk_{T2} 可以由内部时钟源或外部时钟源产生，具体由时钟选择位 CS22:0 确定。没有选择时钟源时 (CS22:0 = 0) 定时器停止。但是不管有没有 clk_{T2}，CPU 都可以访问 TCNT2。CPU 写操作比计数器其他操作 (清零、加减操作) 的优先级高。

计数序列由 T/C 控制寄存器 (TCCR2A) 的 WGM21 和 WGM20 与 TCCR2B 的 WGM22 决定。计数器计数行为与输出比较 OC2A 与 OC2B 的波形有紧密的关系。有关计数序列和波形产生的详细信息请参考 P128” 工作模式 ” 。

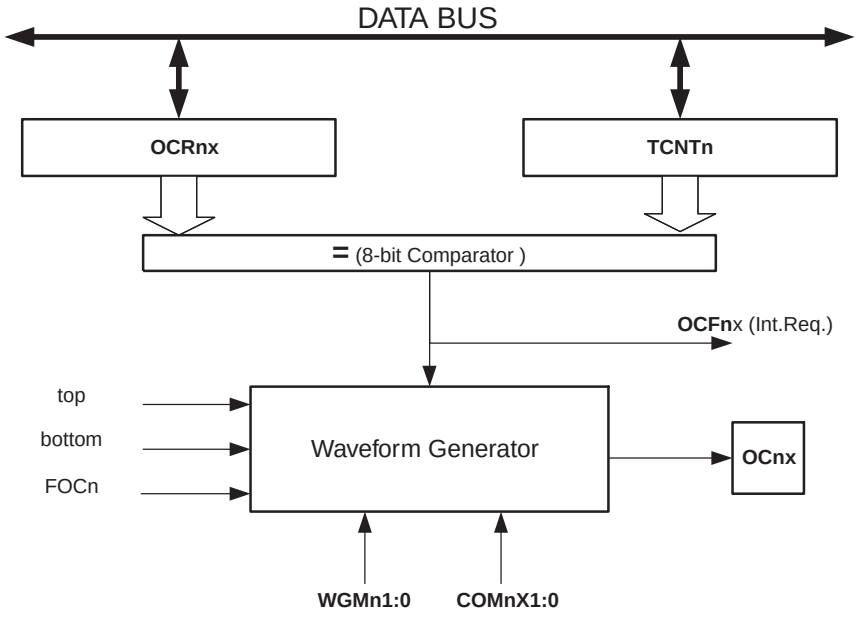
T/C 溢出中断标志 TOV2 根据 WGM22:0 设定的工作模式来设置。TOV2 可以用于产生 CPU 中断。

输出比较单元

8位比较器持续对TCNT2和输出比较匹配寄存器OCR2A与OCR2B进行比较。一旦TCNT2等于 OCR2A 或 OCR2B，比较器就给出匹配信号。在匹配发生的下一个定时器时钟周期里输出比较标志 OCF2A 或 OCF2B 置位。若 OCIE2A = 1 还将引发输出比较中断。执行中断服务程序时 OCF2A 将自动清零，也可以通过软件写 ”1” 的方式进行清零。根据 WGM22:0 和 COM2x1:0 设定的不同工作模式，波形发生器可以利用匹配信号产生不同的波形。同时，波形发生器还利用 max 和 bottom 信号来处理极值条件下的特殊情况 (P128” 工作模式 ”)。

Figure56 为输出比较单元的方框图。

Figure 56. 输出比较单元方框图



使用 PWM 模式时 OCR2x 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式双缓冲功能是禁止的。双缓冲可以将更新 OCR2x 寄存器与 top 或 bottom 时刻同步起来，从而防止产生不对称的 PWM 脉冲，消除毛刺。

访问 OCR2x 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCR2x 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR2x 本身。

强制输出比较

工作于非 PWM 模式时，可以对强制输出比较位 FOC2x 写 ”1” 来产生比较匹配。强制比较匹配不会置位 OCF2x 标志，也不会重载 / 清零定时器，但是 OC2x 引脚将被更新，好象真的发生了比较匹配一样 (COM2x1:0 决定 OC2x 是置位、清零，还是交替变化)。

写 TCNT2 操作阻止比较匹配

CPU 对 TCNT2 寄存器的写操作会在下一个定时器时钟周期阻止比较匹配的发生，即使此时定时器已经停止了。这个特性可以用来将 OCR2x 初始化为与 TCNT2 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT2 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变 TCNT2 就会有风险，不管 T/C 是否在运行。如果写入的 TCNT2 的数值等于 OCR2x，比较匹配就被忽略了，造成不正确的波形发生结果。类似地，在计数器进行降序计数时不要对 TCNT2 写入 BOTTOM。

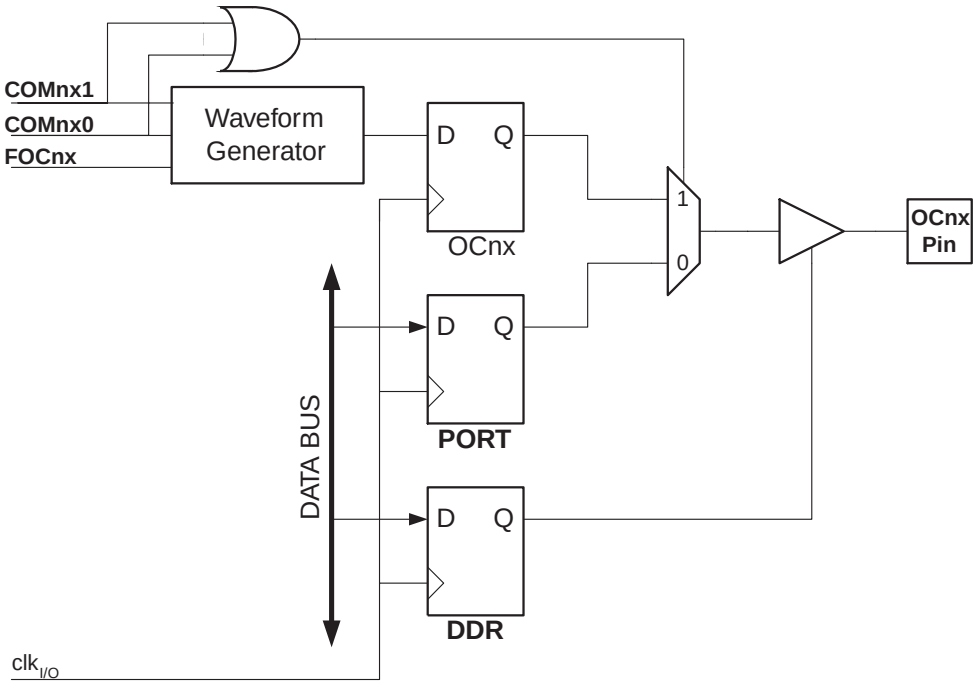
OC2x 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC2x 的方法是在普通模式下利用强制输出比较 FOC2x。即使在改变波形发生模式时 OC2x 寄存器也会一直保持它的数值。

注意 COM2x1:0 和比较数据都不是双缓冲的。 COM2x1:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM2x1:0 具有双重功能。波形发生器利用 COM2x1:0 来确定下一次比较匹配发生时的输出比较状态 (OC2x) ；COM2x1:0 还控制 OC2x 引脚输出信号的来源。Figure57 为受 COM2x1:0 设置影响的逻辑的简化原理图。I/O 寄存器、I/O 位和 I/O 引脚以粗体表示。图中只给出了受 COM2x1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC2x 状态时指的是内部 OC2x 寄存器，而不是 OC2x 引脚。

Figure 57. 比较匹配输出单元原理



只要 COM2x1:0 的一个或两个置位，波形发生器的输出比较功能 OC2x 就会取代通用 I/O 口功能。但是 OC2x 引脚的方向仍然受控于数据方向寄存器 (DDR)。在使用 OC2x 功能之前首先要通过数据方向寄存器的 DDR_OC2x 位将此引脚设置为输出。端口功能与波形发生器的工作模式无关。

输出比较逻辑的设计允许 OC2x 状态在输出之前首先进行初始化。要注意某些 COM2x1:0 设置保留给了其他操作类型，详见 P134“8 位 T/C 寄存器说明 ”。

比较输出模式和波形产生

波形发生器利用 COM2x1:0 的方式在普通、CTC 和 PWM 三种模式下有所区别。对于所有的模式，COM2x1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC2x。非 PWM 模式的比较输出请参见 P134Table64 ；快速 PWM 的比较输出于 P134Table65 ；相位修正 PWM 的比较输出于 P134Table66 。

改变 COM2x1:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC2x 来强制立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM22:0) 及比较输出模式 (COM2x1:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM2x1:0 控制 PWM 输出是否反极性。非 PWM 模式时 COM2x1:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P128“ 比较匹配输出单元 ”)。

具体的时序信息请参考 P132”T/C 时序图 ”。

普通模式

普通模式 (WGM22:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到 8 比特的最大值后 (TOP = 0xFF)，由于数值溢出计数器简单地返回到最小值 0x00 重新开始。在 TCNT0 为零的同一个定时器时钟里 T/C 溢出标志 TOV2 置位。此时 TOV2 有点象第 9 位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV2，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

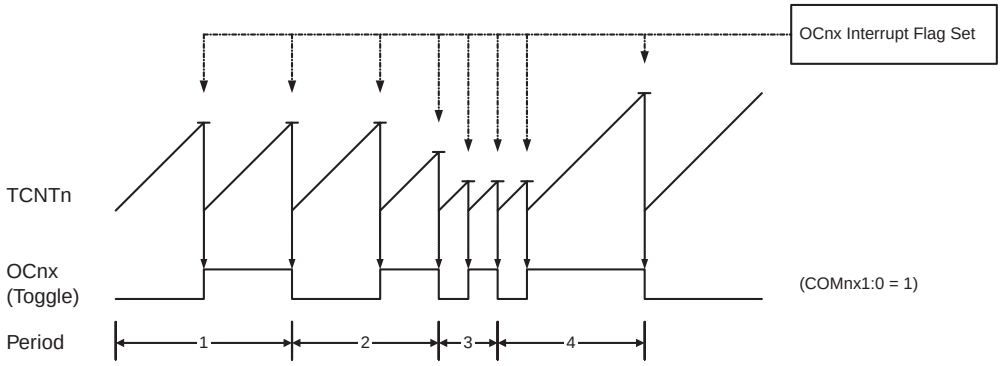
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较产生波形，因为这会占用太多的 CPU 时间。

CTC(比较匹配时清除定时器) 模式

在 CTC 模式 (WGM22:0 = 2) 下 OCR2A 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT2 等于 OCR2A 时计数器清零。OCR2A 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为Figure58。计数器数值TCNT2一直累加到TCNT2与OCR2A匹配，然后 TCNT2 清零。

Figure 58. CTC 模式的时序图



利用 OCF2A 标志可以在计数器数值达到 TOP 即产生中断。在中断服务程序里可以更新TOP的数值。由于CTC模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入 OCR2A 的数值小于当前 TCNT2 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFF，然后再从 0x00 开始计数到 OCR2A。

为了在CTC模式下得到波形输出，可以设置OC2A在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM2A1:0 = 1 来完成。在期望获得 OC2A 输出之前，首先要将其端口设置为输出。波形发生器能够产生的最大频率为 $f_{OC2} = f_{clk_I/O}/2$ (OC2A = 0x00)。频率由如下公式确定：

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

变量 N 代表预分频因子 (1、8、32、 64、 128、256 或 1024)。

在普通模式下， TOV2 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

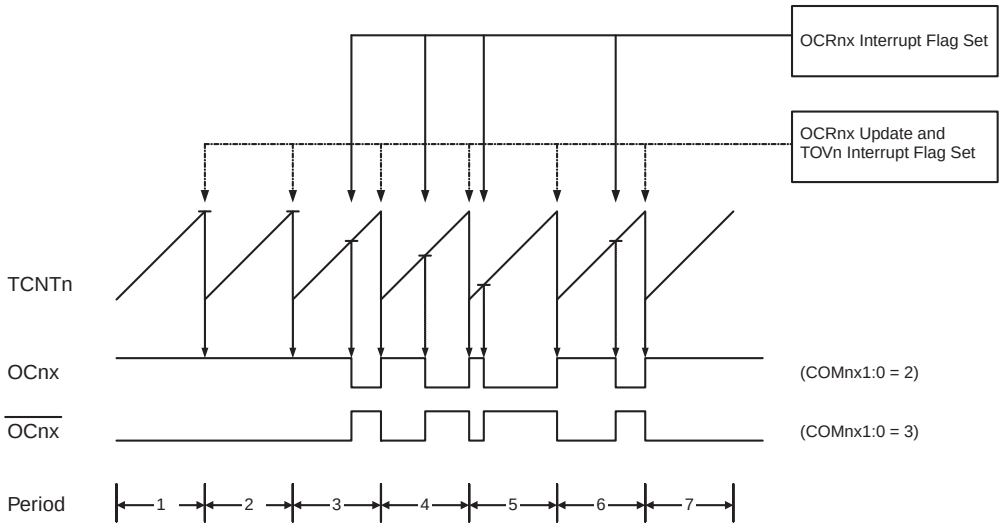
快速 PWM 模式

快速 PWM 模式 (WGM22:0 = 3 或 7) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 MAX，然后立即回到 BOTTOM 重新开始。当 WGM2:0 = 3 时， TOP 值定义为 0xFF ；而当 and OCR2A when MGM2:0 = 7 时，TOP 值为 OCR2A。对于普通的比较输出模式，输出比较引脚 OC2x 在 TCNT2 与 OCR2x 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式，OC2x 的动作正好相反。由于使用了单边斜坡模式，快速 PWM 模式的工作频率比

使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，计数器的数值一直增加到 TOP，然后在后面的一个时钟周期清零。具体的时序图为 Figure59。图中柱状的 TCNT2 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及方向 PWM 输出。TCNT2 斜坡上的短水平线表示 OCR2x 和 TCNT2 的比较匹配。

Figure 59. 快速 PWM 模式时序图



计时器数值达到 TOP 时 T/C 溢出标志 TOV2 置位。如果中断使能，在中断服务程序中断服务程序可以更新比较值。

工作于快速PWM模式时，比较单元可以在OC2x引脚上输出PWM波形。设置COM2x1:0为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形，当 WGM2:0 = 3 时，TOP值定义为0xFF；而当 MGM2:0 = 7时，TOP值为OCR2A(参见 P133Table62)。要想在引脚上得到输出信号还必须将 OC2x 的数据方向设置为输出。产生 PWM 波形的机理是OC2x 寄存器在 OCR2x 与 TCNT2 匹配时置位 (或清零)，以及在计数器清零 (从 TOP 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

变量 N 代表分频因子 (1、8、32、 64、 128、 256 或 1024)。

OCR2A 寄存器为极限值时表示快速 PWM 模式的一些特殊情况。若 OCR2A 等于 BOTTOM，输出为出现在第MAX+1个定时器时钟周期的窄脉冲；OCR2A为MAX时，根据 COM2A1:0 的设定，输出恒为高电平或低电平。

通过设定OC2x在比较匹配时进行逻辑电平取反(COM2x1:0 = 1)，可以得到占空比为50%的周期信号。 OCR2A 为 0 时信号有最高频率 $f_{oc2} = f_{clk_I/O}/2$。这个特性类似于 CTC 模式下的 OC2A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

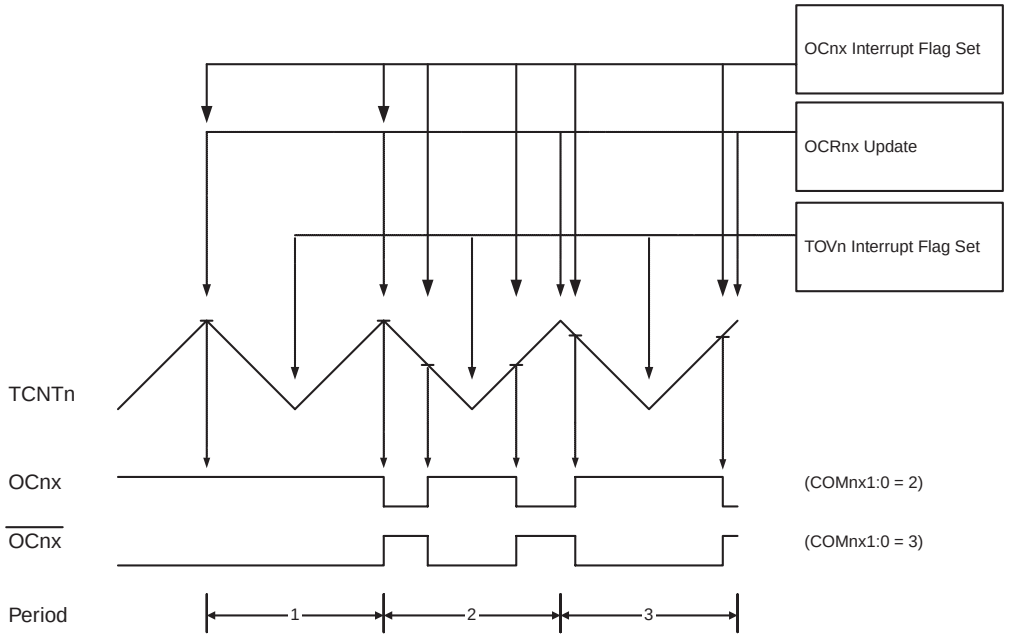
相位修正 PWM 模式

相位修正 PWM 模式 (WGM22:0 = 1 或 5) 为用户提供了一个获得高精度相位修正 PWM 波形的方法。此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。当 WGM2:0 = 3 时， TOP 值定义为 0xFF ；而当 MGM2:0 = 7 时，

TOP值为OCR2A。在一般的比较输出模式下，当计时器往TOP计数时若发生了TCNT2与OCR2x的匹配，OC2x将清零为低电平；而在计时器往BOTTOM计数时若发生了TCNT2于 OCR2x 的匹配， OC2x 将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但由于其对称的特性，十分适合于电机控制。

相位修正 PWM 模式的 PWM 精度固定为 8 比特。计时器不断地累加直到 TOP，然后开始减计数。在一个定时器时钟周期里 TCNT2 的值等于 TOP。时序图可参见 Figure60。图中 TCNT2 的数值用柱状图表示，以说明双斜坡操作。本图同时说明了普通 PWM 的输出和反向 PWM 的输出。 TCNT2 斜坡上的小横条表示 OCR2x 和 TCNT2 的比较匹配。

Figure 60. 相位修正 PWM 模式的时序图



当计时器达到 BOTTOM 时 T/C 溢出标志位 TOV2 置位。此标志位可用来产生中断。

工作于相位修正 PWM 模式时，比较单元可以在 OC2x 引脚产生 PWM 波形：将 COM2x1:0 设置为 2 产生普通相位的 PWM，设置 COM2x1:0 为 3 产生反向 PWM 信号（参见 P133Table63）。要想在引脚上得到输出信号还必须将 OC2x 的数据方向设置为输出。OCR2x 和 TCNT2 比较匹配发生时 OC2x 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由下式公式获得：

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

变量 N 表示预分频因子 (1、8、32、64、 128、 256 或 1024)。

OCR2A 寄存器处于极值代表了相位修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR2A 等于 BOTTOM，输出一直保持为低电平；若 OCR2A 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

在 Figure60 的第 2 个周期，虽然没有发生比较匹配， OCn 也出现了一个从高到低的跳变。其目的是保证波形在BOTTOM两侧的对称。没有比较匹配时有两种情况会出现跳变。

- 如Figure60所示，OCR2A的值从MAX改变为其他数据。当OCR2A值为MAX时，引脚 OCn 的输出应该与前面降序记数比较匹配的结果相同。为保证波形在 BOTTOM

两侧的对称，当 T/C 的数值为 MAX 时，引脚 OCn 的输出又必须符合后面升序记数比较匹配的结果。此时就出现了虽然没有比较匹配发生 OCn 却仍然有跳变的现象。

- 定时器从一个比 OCR2A 大的值开始记数，并因而丢失了一次比较匹配。为了保证波形在 BOTTOM 两侧的对称，此时 OCn 必须立即切换到合适的电平。系统因此引入了第二个虽然没有比较匹配发生 OCn 却仍然有跳变的现象。

T/C 时序图

下面图中给出的 T/C 为同步电路，因此其时钟 clk_{T2} 可以表示为时钟使能信号。在异步模式下，$clk_{I/O}$ 由 T/C 振荡器时钟所取代。图中还说明了中断标志设置的时间。Figure61 包含了 T/C 的基本时序。图中给出了除相位修正 PWM 模式的接近 MAX 值的计数序列。

Figure 61. T/C 时序图，无预分频器

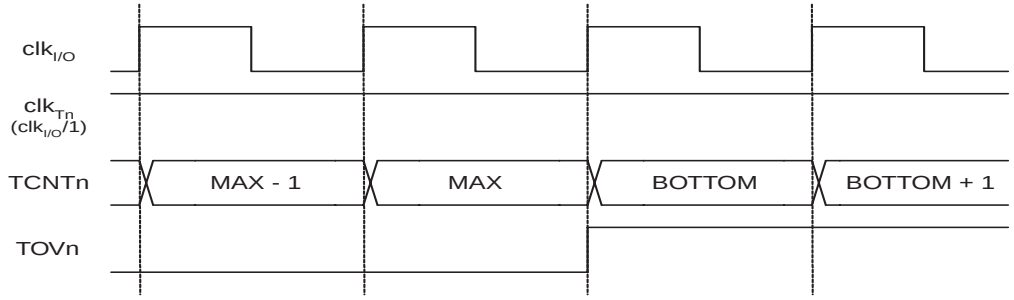


Figure62 给出相同的定时器数据，但预分频器使能。

Figure 62. T/C 时序图，预分频器为 $f_{clk_I/O}/8$

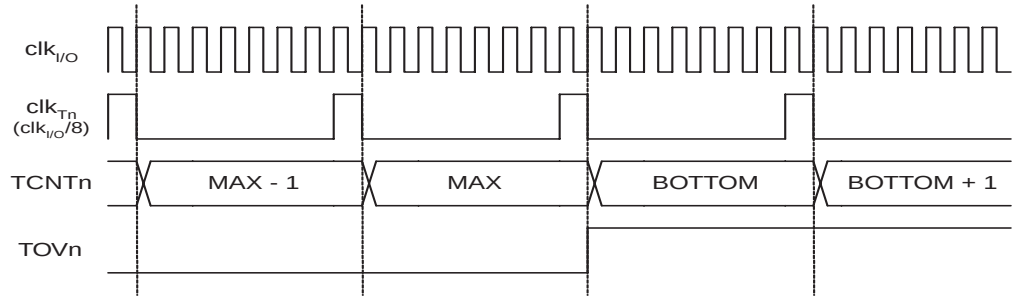


Figure63 给出了各种模式下 (除了 CTC 模式)OCF2A 的置位情况。

Figure 63. T/C 时序图， OCF2A 置位，预分频器为 $f_{clk_I/O}/8$

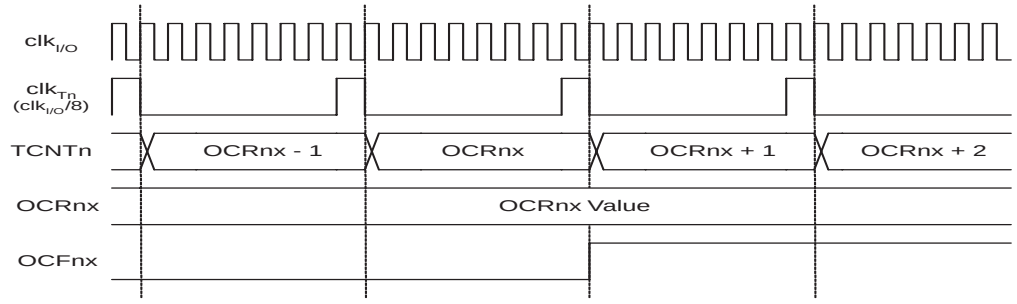
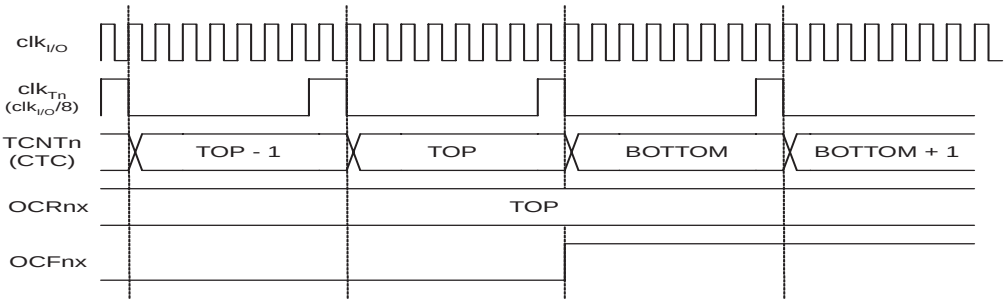


Figure64 给出了 CTC 模式下 OCF2A 置位和 TCNT2 清除的情况。

Figure 64. T/C 时序图， CTC 模式，预分频器为 $f_{clk_I/O}/8$



8 位 T/C 寄存器说明

T/C 控制寄存器 A—TCCR2A

位	7	6	5	4	3	2	1	0	
	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
读 / 写	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7:6 – COM2A1:0: 比较匹配输出 A 模式**

这些位决定了比较匹配发生时输出引脚 OC2A 的电平。如果 COM2A1:0 中的一位或全部都置位，OC2A 以比较匹配输出的方式进行工作。同时其方向控制位要设置为 1 以使能输出驱动。

当 OC2A 连接到物理引脚上时，COM2A1:0 的功能依赖于 WGM22:0 的设置。 Table61 给出了当 WGM22:0 设置为普通模式或 CTC 模式 (非 PWM 模式) 时 COM2A1:0 的功能。

Table 61. 比较输出模式，非 PWM 模式

COM2A1	COM2A0	说明
0	0	正常的端口操作，不与 OC2A 相连接
0	1	比较匹配发生时 OC2A 取反
1	0	比较匹配发生时 OC2A 清零
1	1	比较匹配发生时 OC2A 置位

Table62 给出了当 WGM21:0 设置为快速 PWM 模式时 COM2A1:0 的功能。

Table 62. 比较输出模式，快速 PWM 模式 (1)

COM2A1	COM2A0	说明
0	0	正常的端口操作，不与 OC2A 相连接
0	1	WGM22 = 0：正常的端口操作，不与 OC2A 相连接 WGM22 = 1：比较匹配发生时 OC2A 取反
1	0	比较匹配发生时 OC2A 清零 ，计数到 TOP 时 OC2A 置位
1	1	比较匹配发生时 OC2A 置位 ，计数到 TOP 时 OC2A 清零

Note: 1. 一个特殊情况是OCR2A等于TOP ,且COM2A1置位。此时比较匹配将被忽略 ,而计数到 TOP 时的动作继续有效。详细信息请参见 P129” 快速 PWM 模式 ” 。

Table63 给出了当 WGM22:0 设置为相位修正 PWM 模式时 COM2A1:0 的功能。

Table 63. 比较输出模式，相位修正 PWM 模式 (1)

COM2A1	COM2A0	说明
0	0	正常的端口操作，不与 OC2A 相连接
0	1	WGM22 = 0：正常的端口操作，不与 OC2A 相连接 WGM22 = 1：比较匹配发生时 OC2A 取反
1	0	在升序计数时发生比较匹配将清零 OC2A ；降序计数时发生比较匹配将置位 OC2A
1	1	在升序计数时发生比较匹配将置位 OC2A ；降序计数时发生比较匹配将清零 OC2A

Note: 1. 一个特殊情况是OCR2A等于TOP ,且COM2A1置位。此时比较匹配将被忽略 ,而计数到 TOP 时的动作继续有效。详细信息请参见 P130” 相位修正 PWM 模式 ” 。

- **位 5:4 – COM2B1:0: 比较匹配输出 B 模式**

这些位决定了比较匹配发生时输出引脚 OC2B 的电平。如果 COM2B1:0 中的一位或全部都置位，OC2B 以比较匹配输出的方式进行工作。同时其方向控制位要设置为 1 以使能输出驱动。

当 OC2B 连接到物理引脚上时，COM2B1:0 的功能依赖于 WGM22:0 的设置。 Table64 给出了当 WGM22:0 设置为普通模式或 CTC 模式时 COM2B1:0 的功能。

Table 64. 比较输出模式，非 PWM 模式

COM2B1	COM2B0	说明
0	0	正常的端口操作，不与 OC2B 相连接
0	1	比较匹配发生时 OC2B 取反
1	0	比较匹配发生时 OC2B 清零
1	1	比较匹配发生时 OC2B 置位

Table65 给出了当 WGM22:0 设置为快速 PWM 模式时 COM2B1:0 的功能。

Table 65. 比较输出模式，快速 PWM 模式 (1)

COM2B1	COM2B0	说明
0	0	正常的端口操作，不与 OC2B 相连接
0	1	保留
1	0	比较匹配发生时 OC2B 清零 ，计数到 TOP 时 OC2B 置位
1	1	比较匹配发生时 OC2B 置位 ，计数到 TOP 时 OC2B 清零

Note: 1. 一个特殊情况是OCR2B等于TOP，且COM2B1置位。此时比较匹配将被忽略，而计数到 TOP 时的动作继续有效。详细信息请参见 P130” 相位修正 PWM 模式 ” 。

Table66 给出了当 WGM22:0 设置为相位修正 PWM 模式时 COM2B1:0 的功能。

Table 66. 比较输出模式，相位修正 PWM 模式 (1)

COM2B1	COM2B0	说明
0	0	正常的端口操作，不与 OC2B 相连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC2B ；降序计数时发生比较匹配将置位 OC2B
1	1	在升序计数时发生比较匹配将置位 OC2B ；降序计数时发生比较匹配将清零 OC2B

Note: 1. 一个特殊情况是OCR2B等于TOP，且COM2B1置位。此时比较匹配将被忽略，而计数到 TOP 时的动作继续有效。详细信息请参见 P130” 相位修正 PWM 模式 ” 。

- **位 3, 2 – Res: 保留位**

ATmega48/88/168 中的这些位均为保留位，读返回值为 "0”。

- **位 1:0 – WGM21:0: 波形产生模式**

这两位与 TCCR2B 寄存器的 WGM22 一起控制计数器的计数序列，计数器最大值 TOP 的来源，以及产生何种波形，请见 Table67。 T/C 支持的模式有：普通模式，比较匹配发生时清除计数器模式 (CTC)，以及两种 PWM 模式，详见 P128” 工作模式 ” 。

Table 67. 波形产生模式的位定义

模式	WGM2	WGM1	WGM0	T/C 的工作模式	TOP	OCRx 的更新时间	TOV 标志设置 (1)(2)
0	0	0	0	普通	0xFF	立即更新	MAX
1	0	0	1	相位修正 PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	立即更新	MAX
3	0	1	1	快速 PWM	0xFF	TOP	MAX
4	1	0	0	保留	–	–	–
5	1	0	1	相位修正 PWM	OCRA	TOP	BOTTOM
6	1	1	0	保留	–	–	–
7	1	1	1	快速 PWM	OCRA	TOP	TOP

Notes: 1. MAX= 0xFF
2. BOTTOM= 0x00

T/C 控制寄存器 B—TCCR2B

位	7	6	5	4	3	2	1	0	
	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
读 / 写	W	W	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – FOC2A: 强制输出比较 A**

FOC2A 仅在 WGM 指明非 PWM 模式时才有效。

但是，为了保证与未来器件的兼容性，使用 PWM 时，写 TCCR2B 要对其清零。FOC2A 位写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC2A 将按照 COM2A1:0 的设置输出相应的电平。要注意 FOC2A 类似一个锁存信号，真正对强制输出比较起作用的是 COM2A1:0 的设置。

FOC2A 锁存不会引发任何中断，也不会在使用 OCR2A 作为 TOP 的 CTC 模式下对定时器进行清零。

读 FOC2A 的返回值永远为 "0”。

- **位 6 – FOC2B: 强制输出比较 B**

FOC2B 仅在 WGM 指明非 PWM 模式时才有效。

但是，为了保证与未来器件的兼容性，使用 PWM 时，写 TCCR2B 要对其清零。FOC2B 位写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC2B 将按照 COM2B1:0 的设置输出相应的电平。要注意 FOC2B 类似一个锁存信号，真正对强制输出比较起作用的是 COM2B1:0 的设置。

FOC2B锁存不会引发任何中断，也不会在使用OCR2B作为TOP的CTC模式下对定时器进行清零。

读 FOC2B 的返回值永远为 "0”。

- **位 5:4 – Res: 保留位**

ATmega48/88/168 中的这些位均为保留位，读返回值为 "0”。

- **位 3 – WGM22: 波形产生模式**

见 P134”T/C 控制寄存器 A – TCCR2A” 中的描述。

- **位 2:0 – CS22:0: 时钟选择**

这三个时钟位用于选择 T/C 的时钟源。参见 Table*NOTICE:。

*NOTICE: 时钟选择位定义

CS22	CS21	CS20	说明
0	0	0	无时钟， T/C 不工作
0	0	1	clk_{T2S}/1 (没有预分频)
0	1	0	clk_{T2S}/8 (来自预分频器)
0	1	1	clk_{T2S}/32 (来自预分频器)
1	0	0	clk_{T2S}/64 (来自预分频器)
1	0	1	clk_{T2S}/128 (来自预分频器)
1	1	0	clk_{T2S}/256 (来自预分频器)
1	1	1	clk_{T2S}/1024 (来自预分频器)

如果 T/C0 使用的是外部引脚，那么即使 T0 引脚配置为输出，该引脚的电平变化仍然可以驱动 T/C0。利用这一特性，可用软件控制计数。

T/C 寄存器—TCNT2

位	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT2 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT2 的数值有可能丢失一次 TCNT2 和 OCR2x 的比较匹配。

输出比较寄存器 A—OCR2A

位	7	6	5	4	3	2	1	0	
	OCR2A[7:0]								OCR2A
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 A 包含一个 8 位的数据，不间断地与计数器数值 TCNT2 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC2A 引脚上产生波形。

输出比较寄存器 B—OCR2B

位	7	6	5	4	3	2	1	0	
	OCR2B[7:0]								OCR2B
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 B 包含一个 8 位的数据，不间断地与计数器数值 TCNT2 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC2B 引脚上产生波形。

定时器 / 计数器的异步操作

异步状态寄存器—ASSR

位	7	6	5	4	3	2	1	0	
	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB	ASSR
读 / 写	R	R/W	R/W	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **位 6 – EXCLK: 外部时钟输入使能**

当 EXCLK 为 "1" 且选择了异步时钟，则外部时钟输入缓冲使能，可以从 TOSC1 引脚输入外部时钟，而不是 32 kHz 晶振。 EXCLK 的写操作应在选择异步操作之前完成。只有该位为 "0" 时晶振才能运行。

- **位 5 – AS2: 异步 T/C2**

AS2为"0"时T/C2由I/O时钟$clk_{I/O}$驱动；AS2为"1"时T/C2由连接到TOSC1引脚的晶体振荡器驱动。改变AS2有可能破坏TCNT2、OCR2A、OCR2B、TCCR2A与TCCR2B的内容。

- **位 4 – TCN2UB: T/C2 更新中**

T/C2工作于异步模式时，写TCNT2将引起TCN2UB置位。当TCNT2从暂存寄存器更新完毕后 TCN2UB 由硬件清零。 TCN2UB 为 0 表明 TCNT2 可以写入新值了。

- **位 3 – OCR2AUB: 输出比较寄存器 2 更新中**

T/C2 工作于异步模式时，写OCR2A 将引起 OCR2UB 置位。当 OCR2A 从暂存寄存器更新完毕后 OCR2AUB 由硬件清零。 OCR2AUB 为 0 表明 OCR2A 可以写入新值了。

- **位 2 – OCR2BUB: 输出比较寄存器 2 更新中**

T/C2工作于异步模式时，写OCR2B将引起OCR2BUB置位。当OCR2B从暂存寄存器更新完毕后 OCR2BUB 由硬件清零。 OCR2BUB 为 0 表明 OCR2B 可以写入新值了。

- **位 1 – TCR2AUB: T/C 控制寄存器 2 更新中**

T/C2工作于异步模式时，写TCCR2A将引起TCR2AUB置位。当TCCR2A从暂存寄存器更新完毕后 TCR2AUB 由硬件清零。 TCR2AUB 为 0 表明 TCCR2A 可以写入新值了。

- **位 0 – TCR2BUB: T/C 控制寄存器 2 更新中**

T/C2工作于异步模式时，写TCCR2B将引起TCR2BUB置位。当TCCR2B从暂存寄存器更新完毕后 TCR2BUB 由硬件清零。 TCR2BUB 为 0 表明 TCCR2B 可以写入新值了。

如果在更新忙标志置位的时候写上述任何一个寄存器都将引起数据的破坏，并引发不必要的中断。

读取 TCNT2、OCR2A、OCR2B、TCCR2A 与 TCCR2B 的机制是不同的。读取 TCNT2 得到的是实际的值，而 OCR2A、 OCR2B、 TCCR2A 与 TCCR2B 则是从暂存寄存器中读取的。

定时器 / 计数器 2 的异步操作

T/C2 工作于异步模式时要考虑如下几点：

- 警告 : 在同步和异步模式之间的转换有可能造成 TCNT2、OCR2x 和 TCCR2x 数据的损毁。安全的步骤应该是：
  1. 清零 OCIE2x 和 TOIE2 以关闭 T/C2 的中断
  2. 设置 AS2 以选择合适的时钟源
  3. 对 TCNT2、 OCR2x 和 TCCR2x 写入新的数据

4. 切换到异步模式：等待 TCN2xUB、 OCR2xUB 和 TCR2xUB 清零
5. 清除 T/C2 的中断标志
6. 需要的话使能中断

- 系统主时钟必须比晶振高 4 倍以上。
- 写TCNT2 ,OCR2x和TCCR2x时数据首先送入暂存器 ,两个TOSC1时钟正跳变后才锁存到对应到的寄存器。在数据从暂存器写入目的寄存器之前不能执行新的数据写入操作。 3 个寄存器具有各自独立的暂存器，因此写 TCNT2 并不会干扰 OCR2x 的写操作。异步状态寄存器 ASSR 用来检查数据是否已经写入到目的寄存器。
- 如果要用 T/C2 作为 MCU 省电模式或 ADC 噪声抑制模式的唤醒条件 ,则在 TCNT2，OCR2x和TCCR2x更新结束之前不能进入这些休眠模式 ,否则MCU可能会在T/C2设置生效之前进入休眠模式。这对于用 T/C2 的比较匹配中断唤醒 MCU 尤其重要，因为在更新 OCR2x 或 TCNT2 时比较匹配是禁止的。如果在更新完成之前 (OCR2xUB 为 0)MCU 就进入了休眠模式，那么比较匹配中断永远不会发生， MCU 也永远无法唤醒了。
- 如果要用 T/C2 作为省电模式或 ADC 噪声抑制模式的唤醒条件 ,必须注意重新进入这些休眠模式的过程。中断逻辑需要一个 TOSC1 周期进行复位。如果从唤醒到重新进入休眠的时间小于一个 TOSC1 周期，中断将不再发生，器件也无法唤醒。如果用户怀疑自己程序是否满足这一条件，可以采取如下方法：
  1. 对 TCCR2x、 TCNT2 或 OCR2x 写入合适的数据
  2. 等待 ASSR 相应的更新忙标志清零
  3. 进入省电模式或 ADC 噪声抑制模式
- 若选择了异步工作模式，T/C2 的 32.768 kHz 振荡器将一直工作，除非进入掉电模式或 Standby 模式。用户应该注意，此振荡器的稳定时间可能长达 1 秒钟。因此，建议用户在器件上电复位，或从掉电 /Standby 模式唤醒时至少等待 1 秒钟后再使用 T/C2。同时，由于启动过程时钟的不稳定性，唤醒时所有的 T/C2 寄存器的内容都可能不正确，不论使用的是晶体还是外部时钟信号。用户必须重新给这些寄存器赋值。
- 使用异步时钟时省电模式或 ADC 噪声抑制模式的唤醒过程：中断条件满足后，在下一个定时器时钟唤醒过程启动。也就是说，在处理器可以读取计数器的数值之前计数器至少又累加了一个时钟。唤醒后 MCU 停止 4 个时钟，接着执行中断服务程序。中断服务程序结束之后开始执行 SLEEP 语句之后的程序。
- 从省电模式唤醒之后的短时间内读取 TCNT2 可能返回不正确的数据。因为 TCNT2 是由异步的 TOSC 时钟驱动的，而读取 TCNT2 必须通过一个与内部 I/O 时钟同步的寄存器来完成。同步发生于每个 TOSC1 的上升沿。从省电模式唤醒后 I/O 时钟重新激活，而读到的 TCNT2 数值为进入休眠模式前的值，直到下一个 TOSC1 上升沿的到来。从省电模式唤醒时 TOSC1 的相位是完全不可预测的，而且与唤醒时间有关。因此，读取 TCNT2 的推荐序列为：
  1. 写一个任意数值到 OCR2x 或 TCCR2x
  2. 等待相应的更新忙标志清零。
  3. 读 TCNT2
- 在异步模式下，中断标志的同步需要 3 个处理器周期加一个定时器周期。在处理器可以读取引起中断标志置位的计数器数值之前计数器至少又累加了一个时钟。输出比较引脚的变化与定时器时钟同步，而不是处理器时钟。

定时器 / 计数器 2 中断屏蔽寄存器—TIMSK2

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
读 / 写	R	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 2 – OCIE2B: T/C2 输出比较匹配 B 中断使能**

当 OCIE2B 和状态寄存器的全局中断使能位 I 都为 "1" 时， T/C2 的输出比较匹配 B 中断使能。当 T/C2 的比较匹配发生 ,即 TIFR2 中的 OCF2B 置位时 ,中断服务程序得以执行。

- **位 1 – OCIE2A: 输出比较匹配 A 中断使能**

当 OCIE2A 和状态寄存器的全局中断使能位 I 都为 "1" 时， T/C2 的输出比较匹配 A 中断使能。当 T/C2 的比较匹配发生 ,即 TIFR2 中的 OCF2A 置位时 ,中断服务程序得以执行。

- **位 0 – TOIE2: T/C2 溢出中断使能**

当 TOIE2 和状态寄存器的全局中断使能位 I 都为 "1" 时，T/C2 的溢出中断使能。当 T/C2 发生溢出，即 TIFR2 中的 TOV2 位置位时，中断服务程序得以执行。

定时器 / 计数器 2 中断标志寄存器—TIFR2

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCF2B	OCF2A	TOV2	TIFR2
读 / 写	R	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 2 – OCF2B: 输出比较标志 2 B**

当 T/C2 与 OCR2B(输出比较寄存器 2) 的值匹配时，OCF2B 置位。此位在中断服务程序里硬件清零，也可以通过对其写 1 来清零。当 SREG 中的位 I、 OCIE2B 和 OCF2B 都置位时，中断服务程序得到执行。

- **位 1 – OCF2A: 输出比较标志 2 A**

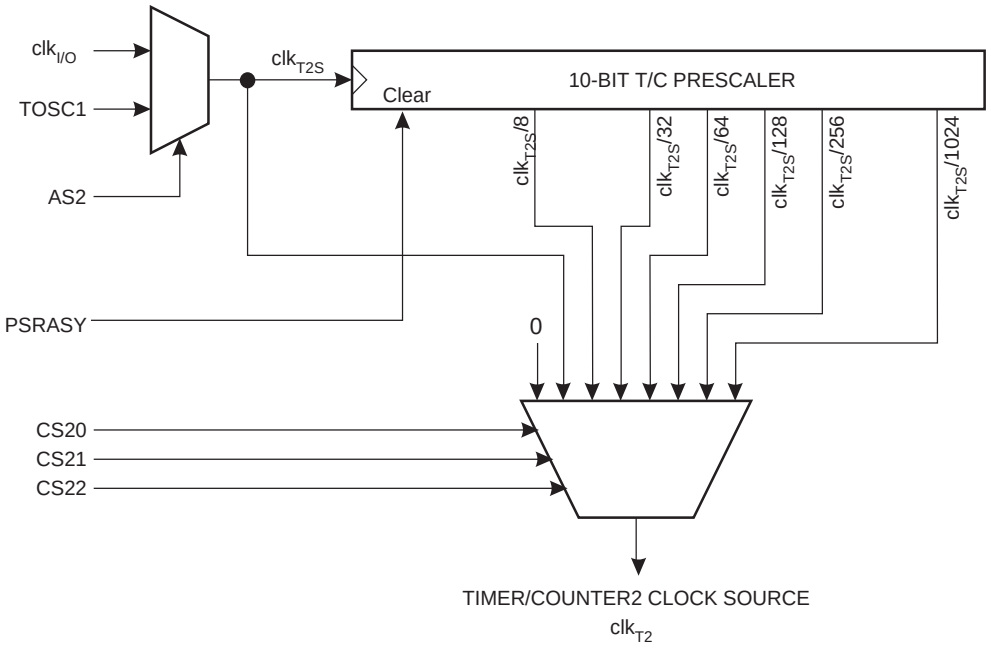
当 T/C2 与 OCR2A(输出比较寄存器 2) 的值匹配时，OCF2A 置位。此位在中断服务程序里硬件清零，也可以通过对其写 1 来清零。当 SREG 中的位 I、 OCIE2A 和 OCF2A 都置位时，中断服务程序得到执行。

- **位 0 – TOV2: T/C2 溢出标志**

当 T/C2 溢出时， TOV2 置位。执行相应的中断服务程序时此位硬件清零。此外， TOV2 也可以通过写 1 来清零。当 SREG 中的位 I、TOIE2A 和 TOV2 都置位时，中断服务程序得到执行。在 PWM 模式中，当 T/C2 在 0x00 改变记数方向时， TOV2 置位。

定时器 / 计数器预分频器

Figure 65. T/C2 的预分频器



T/C2 预分频器的输入时钟称为 clk_{T2S}。缺省地， clk_{T2} 与系统主时钟 $clk_{I/O}$ 连接。若置位 ASSR 的 AS2 , T/C2 将由引脚 TOSC1 异步驱动 , 使得 T/C2 可以作为一个实时时钟 RTC。此时 TOSC1 和 TOSC2 从端口 C 脱离，引脚上可外接一个时钟晶振（内部振荡器针对 32.768 kHz 的钟表晶体进行了优化)。不推荐在 TOSC1 上直接施加外部时钟信号。

T/C2 的可能预分频选项有：clk_{T2S}/8、 clk_{T2S}/32、 clk_{T2S}/64、 clk_{T2S}/128、 clk_{T2S}/256 和 clk_{T2S}/1024。此外还可以选择clk_{T2S}和0 (停止工作)。置位GTCCR寄存器的PSR2将复位预分频器，从而允许用户从可预测的预分频器开始工作。

通用 T/C 控制寄存器—GTCCR

位	7	6	5	4	3	2	1	0	
	TSM	–	–	–	–	–	PSRASY	PSRSYNC	GTCCR
读 / 写	R/W	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1 – PSR2:T/C2 预分频器复位**

此位写 "1“ 时复位 T/C2 的预分频器。操作完成后通常立即由硬件清零。如果 T/C2 工作于异步模式，则这一位置位后一直保持到预分频器复位操作真正完成。如果 TSM 位置位，则该位不会被硬件清零。对T/C同步模式的描述请参见 P95”位7 – TSM: T/C 同步模式”。

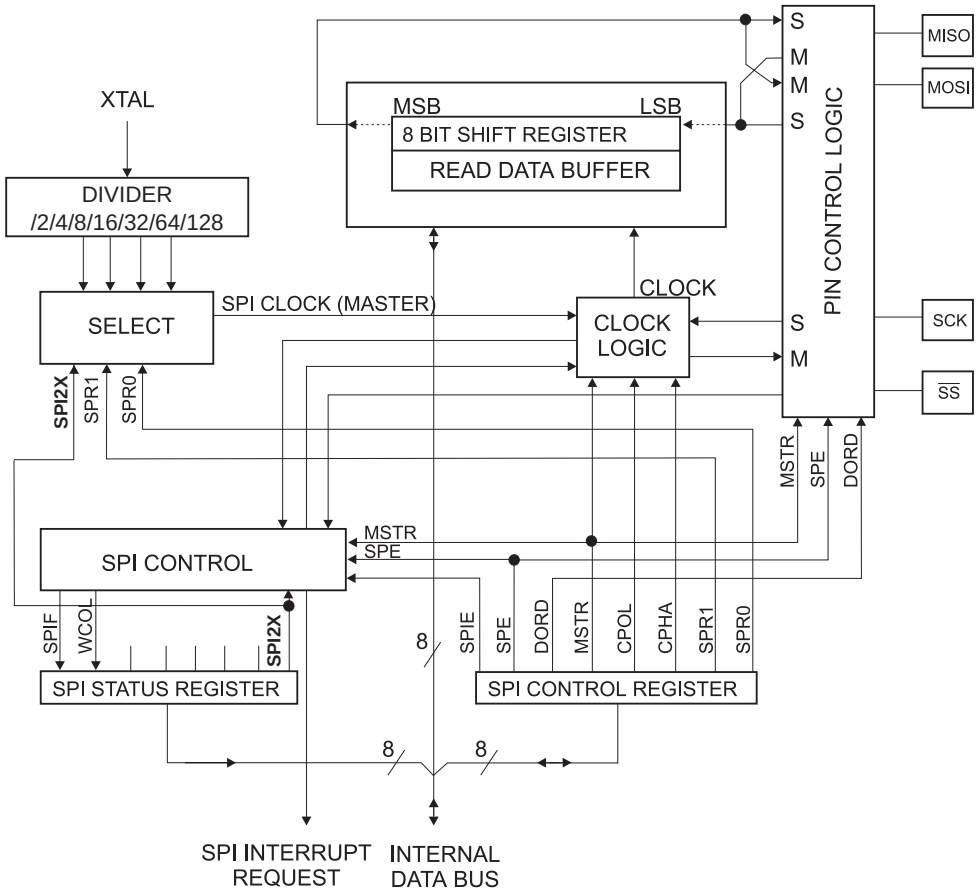
串行外设接口—SPI

串行外设接口 SPI 允许 ATmega48/88/168 和外设或其他 AVR 器件进行高速的同步数据传输。 ATmega48/88/168 SPI 的特点如下：

- **全双工，3 线同步数据传输**
- **主机或从机操作**
- **LSB 首先发送或 MSB 首先发送**
- **7 种可编程的比特率**
- **传输结束中断标志**
- **写碰撞标志检测**
- **可以从闲置模式唤醒**
- **作为主机时具有倍速模式 (CK/2)**

USART 也可用于 SPI 主机模式，见 P177 “SPI 模式下的 USART” 。

Figure 66. SPI 方框图 (1)



Note: 1. SPI 的引脚排列请参见 P2Figure 1 与 P65Table34 。

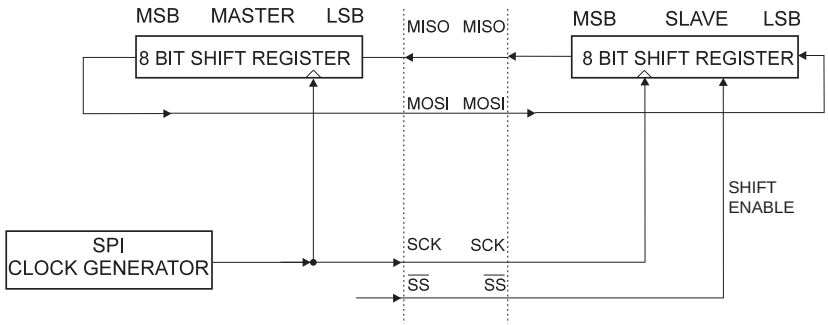
主机和从机之间的 SPI 连接如 Figure67 所示。系统包括两个移位寄存器和一个主机时钟发生器。通过将需要的从机的 $\overline{SS}$ 引脚拉低，主机启动一次通讯过程。主机和从机将需要发送的数据放入相应的移位寄存器。主机在 SCK 引脚上产生时钟脉冲以交换数据。主机的数据从主机的 MOSI 移出，从从机的 MOSI 移入；从机的数据从从机的 MISO 移出，从主机的 MISO 移入。主机通过将从机的 $\overline{SS}$ 拉高实现与从机的同步。

配置为 SPI 主机时，SPI 接口不自动控制 $\overline{SS}$ 引脚，必须由用户软件在通信开始前进行处理。对 SPI 数据寄存器写入数据即启动 SPI 时钟，将 8 比特的数据移入从机。传输结束后 SPI 时钟停止，传输结束标志 SPIF 置位。如果此时 SPCR 寄存器的 SPI 中断使能位

SPIE置位，中断就会发生。主机可以继续往SPDR写入数据以移位到从机中去，或者是将从机的 $\overline{SS}$ 拉高以说明数据包发送完成。最后进来的数据将一直保存于缓冲寄存器里。

配置为从机时，只要 $\overline{SS}$ 为高，SPI 接口将一直保持睡眠状态，并保持 MISO 为三态。在这个状态下软件可以更新 SPI 数据寄存器 SPDR 的内容。即使此时 SCK 引脚有输入时钟，SPDR 的数据也不会移出，直至 $\overline{SS}$ 被拉低。一个字节完全移出之后，传输结束标志SPIF置位。如果此时SPCR寄存器的SPI中断使能位SPIE置位，就会产生中断请求。在读取移入的数据之前从机可以继续往 SPDR 写入数据。最后进来的数据将一直保存于缓冲寄存器里。

Figure 67. SPI 主机 - 从机的互连



SPI 系统的发送方向只有一个缓冲器，而在接收方向有两个缓冲器。也就是说，在发送时一定要等到移位过程全部结束后才能对 SPI 数据寄存器执行写操作。而在接收数据时，需要在下一个字符移位过程结束之前通过访问SPI 数据寄存器读取当前接收到的字符。否则第一个字节将丢失。

工作于 SPI 从机模式时，控制逻辑对 SCK 引脚的输入信号进行采样。为了保证对时钟信号的正确采样， SPI 时钟不能超过 $f_{osc}/4$

SPI 使能后，MOSI、MISO、SCK 和 $\overline{SS}$ 引脚的数据方向将按照 Table68 所示自动进行配置。更多自动重载信息请参考 P64” 端口的第二功能 ”。

Table 68. SPI 引脚重载 (1)

引脚	方向，SPI 主机	方向，SPI 从机
MOSI	用户定义	输入
MISO	输入	用户定义
SCK	用户定义	输入
$\overline{SS}$	用户定义	输入

Note: 1. 请参考 P66”端口 B 的第二功能”以了解如何定义由用户定义的 SPI 引脚。

下面的例程说明如何将 SPI 初始化为主机，以及如何进行简单的数据发送。例子中 DDR_SPI必须由实际的数据方向寄存器代替；DD_MOSI、DD_MISO和DD_SCK必须由实际的数据方向代替。比如说，MOSI 为 PB5 引脚，则 DD_MOSI 要用 DDB5 取代，DDR_SPI 则用 DDRB 取代。

汇编代码例程 (1)

```
SPI_MasterInit:
  ; 设置 MOSI 和 SCK 为输出，其他为输入
  ldi   r17,(1<<DD_MOSI)|(1<<DD_SCK)
  out   DDR_SPI,r17
  ; 使能 SPI 主机模式，设置时钟速率为 fck/16
  ldi   r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
  out   SPCR,r17
  ret

SPI_MasterTransmit:
  ; 启动数据传输 (r16)
  out   SPDR,r16
Wait_Transmit:
  ; 等待传输结束
  sbis  SPSR,SPIF
  rjmp  Wait_Transmit
  ret
```

C 代码例程 (1)

```
void SPI_MasterInit(void)
{
  /* 设置 MOSI 和 SCK 为输出，其他为输入 */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* 使能 SPI 主机模式，设置时钟速率为 fck/16*/
  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* 启动数据传输 */
  SPDR = cData;
  /* 等待传输结束 */
  while(!(SPSR & (1<<SPIF)))
    ;
}
```

Note: 1. 程序假定已经包含了正确的头文件。

下面的例子说明如何将 SPI 初始化为从机，以及如何进行简单的数据接收。

汇编代码例程 (1)

```
SPI_SlaveInit:
  ; 设置 MISO 为输出，其他为输入
  ldi   r17,(1<<DD_MISO)
  out   DDR_SPI,r17
  ; 使能 SPI
  ldi   r17,(1<<SPE)
  out   SPCR,r17
  ret

SPI_SlaveReceive:
  ; 等待接收结束
  sbis  SPSR,SPIF
  rjmp  SPI_SlaveReceive
  ; 读取接收到的数据，然后返回
  in    r16,SPDR
  ret
```

C 代码例程 (1)

```
void SPI_SlaveInit(void)
{
  /* 设置 MISO 为输出，其他为输入 */
  DDR_SPI = (1<<DD_MISO);
  /* 使能 SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* 等待接收结束 */
  while(!(SPSR & (1<<SPIF)))
    ;
  /* 返回数据 */
  return SPDR;
}
```

Note: 1. 例程假定已经包含了正确的头文件。

$\overline{SS}$ 引脚的功能

从机模式

SPI 配置为从机时，从机选择引脚 $\overline{SS}$ 总是为输入。$\overline{SS}$ 为低将激活 SPI 接口，MISO 成为输出（用户必须进行相应的端口配置）引脚，其他引脚成为输入引脚。当 $\overline{SS}$ 为高时所有的引脚成为输入， SPI 逻辑复位，不再接收数据。

$\overline{SS}$ 引脚对于数据包 / 字节的同步非常有用，可以使从机的位计数器与主机的时钟发生器同步。当 $\overline{SS}$ 拉高时 SPI 从机立即复位接收和发送逻辑，并丢弃移位寄存器里不完整的数据。

主机模式

当 SPI 配置为主机时 (SPCR 的 MSTR 置位)，用户可以决定 $\overline{SS}$ 引脚的方向。

若 $\overline{SS}$ 配置为输出，则此引脚可以用作普通的 I/O 口而不影响 SPI 系统。典型应用是用来驱动从机的 $\overline{SS}$ 引脚。

如果 $\overline{SS}$ 配置为输入，必须保持为高以保证 SPI 的正常工作。若系统配置为主机， $\overline{SS}$ 为输入，但被外设拉低，则 SPI 系统会将此低电平解释为有一个外部主机将自己选择为从机。为了防止总线冲突， SPI 系统将实现如下动作：

1. 清零 SPCR 的 MSTR 位，使 SPI 成为从机，从而 MOSI 和 SCK 变为输入。
2. SPSR 的 SPIF 置位。若 SPI 中断和全局中断开放，则中断服务程序将得到执行。

因此，使用中断方式处理 SPI 主机的数据传输，并且存在 $\overline{SS}$ 被拉低的可能性时，中断服务程序应该检查 MSTR 是否为 "1”。若被清零，用户必须将其置位，以重新使能 SPI 主机模式。

SPI 控制寄存器—SPCR

位	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – SPIE: SPI 中断使能**

置位后，只要 SPSR 寄存器的 SPIF 和 SREG 寄存器的全局中断使能位置位，就会引发 SPI 中断。

- **位 6 – SPE: SPI 使能**

SPE 置位将使能 SPI。进行任何 SPI 操作之前必须置位 SPE。

- **位 5 – DORD: 数据次序**

DORD 置位时数据的 LSB 首先发送；否则数据的 MSB 首先发送。

- **位 4 – MSTR: 主 / 从选择**

MSTR 置位时选择主机模式，否则为从机。如果 MSTR 为 "1”，$\overline{SS}$ 配置为输入，但被拉低，则 MSTR 被清零，寄存器 SPSR 的 SPIF 置位。用户必须重新设置 MSTR 进入主机模式。

- **位 3 – CPOL: 时钟极性**

CPOL 置位表示空闲时 SCK 为高电平；否则空闲时 SCK 为低电平。请参考 Figure68 与 Figure69。CPOL 功能总结如下：

Table 69. CPOL 功能

CPOL	起始沿	结束沿
0	上升沿	下降沿
1	下降沿	上升沿

- **位 2 – CPHA: 时钟相位**

CPHA 决定数据是在 SCK 的起始沿采样还是在 SCK 的结束沿采样。请参考 Figure68 与 Figure69 。 CPHA 功能总结如下：

Table 70. CPHA 功能

CPHA	起始沿	结束沿
0	采样	设置
1	设置	采样

- **位 1, 0 – SPR1, SPR0: SPI 时钟速率选择**

确定主机的 SCK 速率。SPR1 和 SPR0 对从机没有影响。SCK 和振荡器的时钟频率 f_{osc} 关系如下表所示：

Table 71. SCK 和振荡器频率的关系

SPI2X	SPR1	SPR0	SCK 频率
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPI 状态寄存器—SPSR

位	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
读 / 写	R	R	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – SPIF: SPI 中断标志**

串行发送结束后，SPIF 置位。若此时寄存器 SPCR 的 SPIE 和全局中断使能位置位，SPI 中断即产生。如果 SPI 为主机， $\overline{SS}$ 配置为输入，且被拉低， SPIF 也将置位。进入中断服务程序后SPIF自动清零。或者可以通过先读SPSR，紧接着访问SPDR来对SPIF清零。

- **位 6 – WCOL: 写碰撞标志**

在发送当中对 SPI 数据寄存器 SPDR 写数据将置位 WCOL。WCOL 可以通过先读 SPSR，紧接着访问 SPDR 来清零。

- **位 5..1 – Res: 保留位**

ATmega48/88/168 的保留位，读操作返回值为 0。

- **位 0 – SPI2X: SPI 倍速**

置位后 SPI 的速度加倍 (见 Table71)。若为主机，则 SCK 频率可达 CPU 频率的一半。若为从机，必须保证此时钟不大于 $f_{osc}/4$ 以保证正常工作。

ATmega48/88/168 的 SPI 接口同时还用来实现程序和 EEPROM 的下载和上载。请参见 P270 的 SPI 串行编程和校验。

SPI 数据寄存器—SPDR

位	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	X	X	X	X	X	X	X	X	未定义

SPI 数据寄存器为读/写寄存器，用来在寄存器文件和SPI移位寄存器之间传输数据。写寄存器将启动数据传输，读寄存器将读取寄存器的接收缓冲器。

数据模式

SCK 的相位、极性与数据间有 4 种组合。CPHA 和 CPOL 控制组合的方式。SPI 数据传输格式见 Figure68 与 Figure69。每一位数据的移出和移入发生于 SCK 不同的信号跳变沿，以保证有足够的时间使数据稳定。这个过程在 Table69 和 Table70 有清楚的说明：

Table 72. CPOL 功能

	起始沿	结束沿	SPI 模式
CPOL=0, CPHA=0	采样 (上升沿)	设置 (下降沿)	0
CPOL=0, CPHA=1	设置 (上升沿)	采样 (下降沿)	1
CPOL=1, CPHA=0	采样 (下降沿)	设置 (上升沿)	2
CPOL=1, CPHA=1	设置 (下降沿)	采样 (上升沿)	3

Figure 68. CPHA = 0 时 SPI 的传输格式

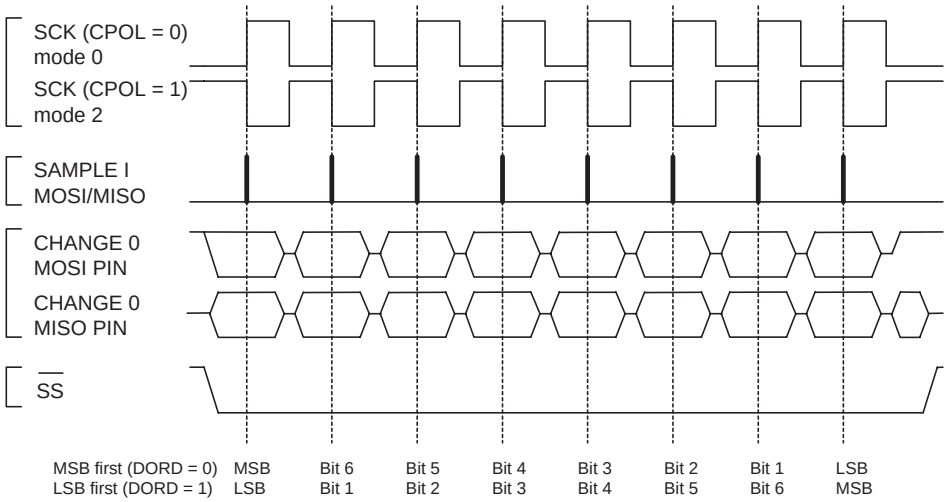
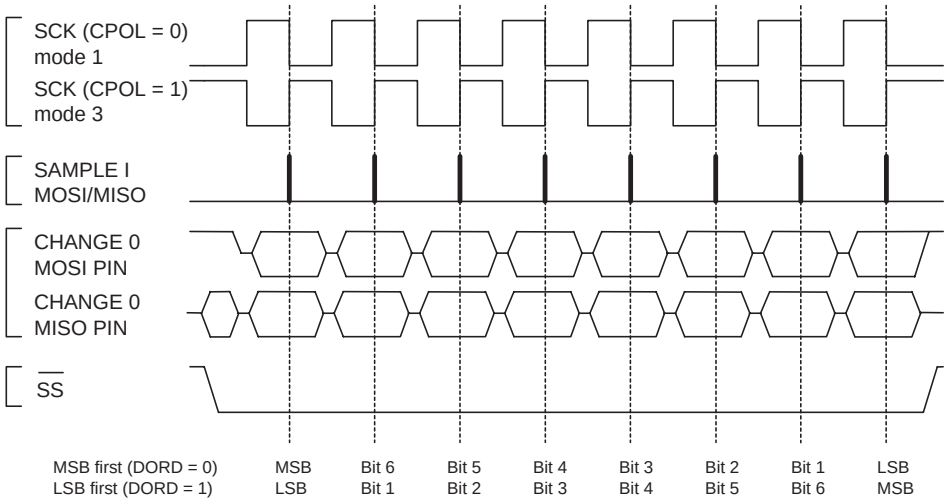


Figure 69. CPHA = 1 时 SPI 的传输格式



USART0

通用同步和异步串行接收器和发送器 (USART) 是一个高度灵活的串行通讯设备。主要特点有：

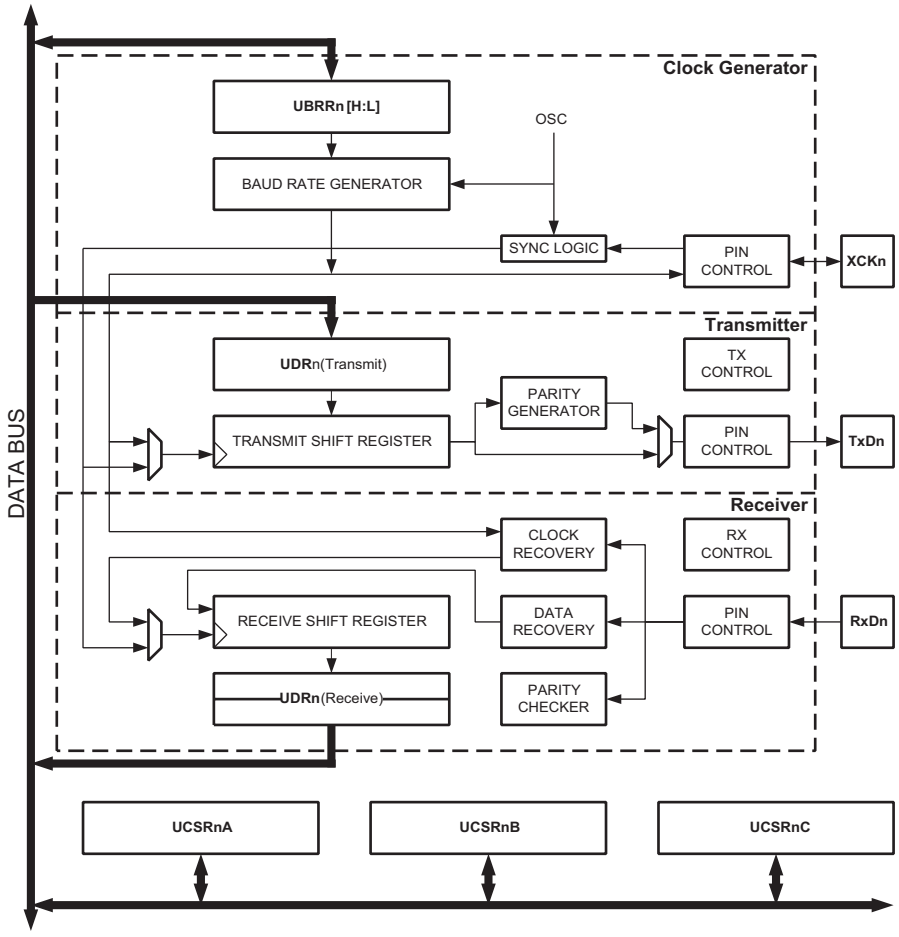
- **全双工操作 (独立的串行接收和发送寄存器)**
- **异步或同步操作**
- **主机或从机提供时钟的同步操作**
- **高精度的波特率发生器**
- **支持 5, 6, 7, 8, 或 9 个数据位和 1 个或 2 个停止位**
- **硬件支持的奇偶校验操作**
- **过高数据率检测**
- **帧错误检测**
- **噪声滤波，包括错误的起始位检测，以及数字低通滤波器**
- **三个独立的中断：发送结束中断，发送数据寄存器空中断，以及接收结束中断**
- **多处理器通讯模式**
- **倍速异步通讯模式**

USART 还可用于主 SPI 模式，请参见 P177 “SPI 模式下的 USART” .

综述

Figure70为USART转发器的简化框图。CPU可以访问的I/O寄存器和I/O引脚以粗体表示。

Figure 70. USART 方框图 (1)



Note: 1. 请参考 P2Figure 1 与 P71Table40 了解 USART0 的引脚分布。

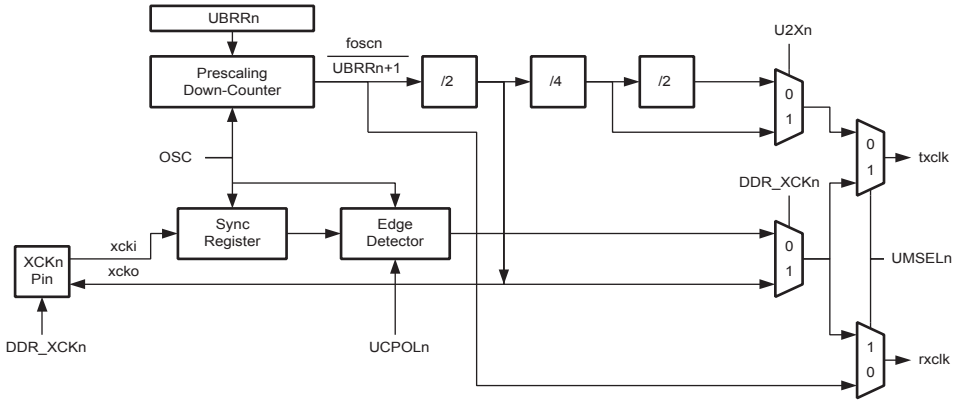
虚线框将 USART 分成了三个主要部分（从上到下）：时钟发生器，发送器和接收器。控制寄存器由三个单元共享。时钟发生器包含同步逻辑，通过它将波特率发生器及为从机同步操作所使用的外部输入时钟同步起来。 XCK（发送器时钟）引脚只用于同步传输模式。发送器包括一个写缓冲器，串行移位寄存器，奇偶发生器以及处理不同的帧格式所需的控制逻辑。写缓冲器可以保持连续发送数据而不会在数据帧之间引入延迟。由于接收器具有时钟和数据恢复单元，它是 USART 模块中最复杂的。恢复单元用于异步数据的接收。除了恢复单元，接收器还包括奇偶校验，控制逻辑，移位寄存器和一个两级接收缓冲器 UDRn。接收器支持与发送器相同的帧格式，而且可以检测帧错误，数据过速和奇偶校验错误。

时钟产生

时钟产生逻辑为发送器和接收器产生基础时钟。USART 支持 4 种模式的时钟：正常的异步模式，倍速的异步模式，主机同步模式，以及从机同步模式。USART 控制位 UMSELn 和状态寄存器 C (UCSRnC) 用于选择异步模式和同步模式。倍速模式（只适用于异步模式）受控于 UCSRnA 寄存器的 U2X。使用同步模式 (UMSELn = 1) 时，XCK 的数据方向寄存器 (DDR_XCKn) 决定时钟源是由内部产生（主机模式）还是由外部生产（从机模式）。仅在同步模式下 XCKn 有效。

Figure71 为时钟产生逻辑的框图。

Figure 71. 时钟产生逻辑框图



信号说明：

txclk 发送器时钟（内部信号）

rxclk 接收器基本时钟（内部信号）

xcki XCK 引脚输入（内部信号），用于同步从机操作

xcko 输出到 XCK 引脚的时钟（内部信号），用于同步主机操作

fosc XTAL 频率（系统时钟）

片内时钟产生—波特率发生器

内部时钟用于异步模式与同步主机模式，请参见 Figure71。

USART的波特率寄存器UBRRn和降序计数器相连接，一起构成可编程的预分频器或波特率发生器。降序计数器对系统时钟计数，当其计数到零或 UBRRnL 寄存器被写时，会自动装入 UBRRn 寄存器的值。当计数到零时产生一个时钟，该时钟作为波特率发生器的输出时钟，输出时钟的频率为 f_{osc}/(UBRRn+1)。发送器对波特率发生器的输出时钟进行 2、8 或 16 的分频，具体情况取决于工作模式。波特率发生器的输出被直接用于接收器与数据恢复单元。数据恢复单元使用了一个有 2、8 或 16 个状态的状态机，具体状态数由 UMSELn、U2Xn 与 DDR_XCKn 位设定的工作模式决定。

Table73 给出了计算波特率(位/秒)以及计算每一种使用内部时钟源工作模式的UBRRn值的公式。

Table 73. 波特率计算公式

工作模式	波特率的计算公式 (1)	UBRRn 值的计算公式
异步正常模式 (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
异步倍速模式 (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
同步主机模式	$BAUD = \frac{f_{OSC}}{2(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. 波特率定义为每秒位的传输速率 (bps)

BAUD 波特率 (bps)

$\mathbf{f_{osc}}$ 系统时钟频率

UBRRn UBRRnH 与 UBRRnL 的数值 (0-4095)

Table81 给出了在某些系统时钟频率下对应的 UBRRn 数值。

倍速工作模式 (U2Xn)

通过设定 UCSRnA 寄存器的 U2Xn 位可以使传输速率加倍。该位只对异步工作模式有效。工作在同步模式时将该位设置为 "0"。

U2Xn置位会把波特率分频器的分频值从16降到8 ,使异步通信的传输速率加倍。此时接收器只使用一半的采样数对数据进行采样及时钟恢复，因此在该模式下需要更精确的系统时钟与波特率设置。发送器则没有这个要求。

外部时钟

同步从机操作模式由外部时钟驱动，如 Figure71 所示。

从 XCKn 引脚输入的外部时钟由同步寄存器进行采样 ,用以提高稳定性。同步寄存器的输出先通过一个边沿检测器，然后才能够为发送器与接收器所使用。这一过程引入了两个 CPU 时钟周期的延时，因此外部 XCK 的最大时钟频率由以下公式决定：

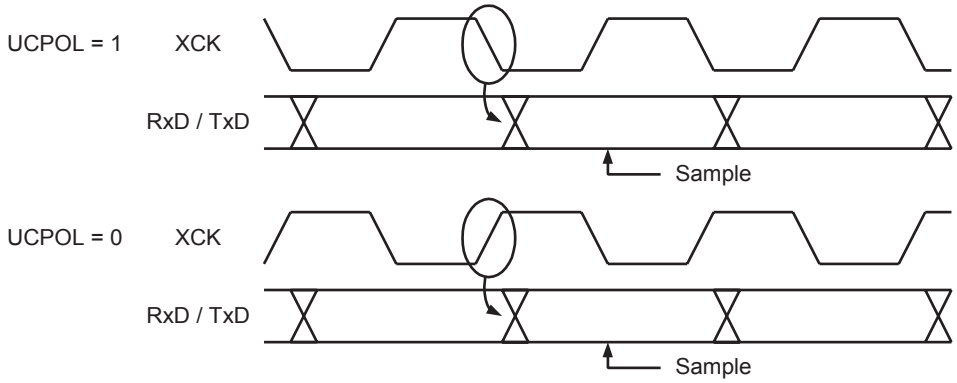
$$f_{XCK} < \frac{f_{OSC}}{4}$$

要注意 f_{osc} 由系统时钟的稳定性决定，为了防止因频率漂移而丢失数据，建议保留足够的裕量。

同步时钟操作

使用同步模式时 (UMSELn = 1) ,XCKn 引脚被用于时钟输入 (从机模式) 或时钟输出 (主机模式)。时钟的边沿、数据的采样与数据的变化之间的关系的基本规律是：在改变数据输出端 TxDn 的 XCKn 时钟的相反边沿对数据输入端 RxDn 进行采样。

Figure 72. 同步模式时的 XCK 时序



UCRSC 寄存器中的 UCPOLn 确定使用 XCKn 时钟的哪个边沿对数据采样和改变输出数据。如 Figure72 所示，当 UCPOLn=0 时，在 XCKn 的上升沿改变输出数据，在 XCKn 的下降沿进行数据采样；当 UCPOLn=1 时，则相反。

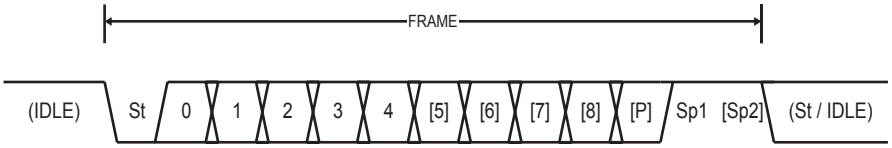
帧格式

串行数据帧由数据字加上同步位 (开始位与停止位) 以及用于纠错的奇偶校验位构成。USART 接受以下 30 种组合的数据帧格式：

- 1 个起始位
- 5、6、7、8 或 9 个数据位
- 无校验位、奇校验或偶校验位
- 1 或 2 个停止位

数据帧以起始位开始，紧接着是数据字的最低位，数据字最多可以有 9 个数据位，以数据的最高位结束。如果使能了校验位，校验位将紧接着数据位，最后是结束位。当一个完整的数据帧传输结束后，可以立即传输下一个新的数据帧，或使传输线处于空闲状态。Figure73 所示为可能的数据帧结构组合。括号中的位是可选的。

Figure 73. 帧格式



St 起始位，总是为低电平

(n) 数据位 (0 ~ 8)

P 校验位，可以为奇校验或偶校验

Sp 停止位，总是为高电平

IDLE 通讯线上没有数据传输 (RxDn 或 TxDn)，线路空闲时必须为高电平

数据帧的结构由 UCSRnB 和 UCSRnC 寄存器的 UCSZn2:0、UPMn1:0 与 USBSn 位设定。接收与发送使用相同的设置。设置的任何改变都可能破坏正在进行的数据传送与接收。

USART的字长位UCSZn2:0确定了数据帧的数据位数；校验模式位UPMn1:0用于使能与决定校验的类型； USBSn 位设置一位或两位结束位。接收器忽略第二个停止位，因此帧错误 (FE) 只在第一个结束位为 "0” 时被检测到。

校验位的计算

校验位的计算是对数据的各个位进行异或运算。如果选择了奇校验，则异或结果还需要取反。校验位与数据位的关系如下：

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even} 偶校验结果

P_{odd} 奇校验位结果

d_n 第 n 个数据位

校验位处于最后一个数据位与第一个停止位之间。

USART 的初始化

进行通信之前首先要对 USART 进行初始化。初始化过程通常包括波特率的设定，帧结构的设定，以及根据需要使能接收器或发送器。对于中断驱动的 USART 操作，在初始化时首先要清零全局中断标志位 (全局中断被屏蔽)。

重新改变 USART 的设置应该在没有数据传输的情况下进行。TXCn 标志位可以用来检验一个数据帧的发送是否已经完成， RXC 标志位可以用来检验接收缓冲器中是否还有数据未读出。在每次发送数据之前 (在写发送数据寄存器 UDR 之前)TXCn 标志位必须清零。

以下是 USART 初始化程序示例。例程采用了轮询（中断被禁用）的异步操作，而且帧结构是固定的。波特率作为函数参数给出。在汇编程序里波特率参数保存于寄存器 r17:r16。

汇编代码例程 (1)

```
USART_Init:
  ; 设置波特率
  out   UBRRnH, r17
  out   UBRRnL, r16
  ; 接收器与发送器使能
  ldi   r16, (1<<RXENn)|(1<<TXENn)
  out   UCSRnB,r16
  ; 设置帧格式： 8 个数据位， 2 个停止位
  ldi   r16, (1<<USBSn)|(3<<UCSZn0)
  out   UCSRnC,r16
  ret
```

C 代码例程 (1)

```
void USART_Init( unsigned int baud )
{
  /* 设置波特率*/
  UBRRnH = (unsigned char)(baud>>8);
  UBRRnL = (unsigned char)baud;
  /* 接收器与发送器使能*/
  UCSRnB = (1<<RXENn)|(1<<TXENn);
  /* 设置帧格式： 8 个数据位， 2 个停止位 */
  UCSRnC = (1<<USBSn)|(3<<UCSZn0);
}
```

Note: 1. 本代码假定已经包含了合适的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

更高级的初始化程序可将帧格式作为参数、禁止中断等。然而许多应用程序使用固定的波特率与控制寄存器。此时初始化代码可以直接放在主程序中，或与其它 I/O 模块的初始化代码组合到一起。

数据发送—USART 发送器

置位 UCSRnB 寄存器的发送允许位 TXEN 将使能 USART 的数据发送。使能后 TxDn 引脚的通用 I/O 功能即被 USART 功能所取代，成为发送器的串行输出引脚。发送数据之前要设置好波特率、工作模式与帧结构。使用同步发送模式时，施加于 XCKn 引脚的时钟信号即为数据发送的时钟。

发送 5 到 8 位数据位的帧

将需要发送的数据加载到发送缓存器将启动数据发送。加载过程即为 CPU 对 UDRn 寄存器的写操作。当移位寄存器可以发送新一帧数据时，缓冲的数据将转移到移位寄存器。当移位寄存器处于空闲状态（没有正在进行的数据传输），或前一帧数据的最后一个停止位传送结束，它将加载新的数据。一旦移位寄存器加载了新的数据，就会按照设定的波特率、 U2Xn 位或工作模式所决定的 XCKn 来完成数据的发送。

以下程序给出对 UDREn 标志采用轮询方式发送数据的例子。当发送的数据少于 8 位时，写入 UDR 相应位置的高几位将被忽略。当然，执行本段代码之前首先要初始化 USART。在汇编代码中要发送的数据存放于 R16。

汇编代码例程 (1)

```
USART_Transmit:
  ; 等待发送缓冲器为空
  sbis  UCSRnA,UDREn
  rjmp  USART_Transmit
  ; 将数据放入缓冲器，发送数据
  out   UDRn,r16
  ret
```

C 代码例程 (1)

```
void USART_Transmit( unsigned char data )
{
  /* 等待发送缓冲器为空 */
  while ( !( UCSRnA & (1<<UDREn)) )
        ;
  /* 将数据放入缓冲器，发送数据 */
  UDRn = data;
}
```

Note: 1. 本代码假定已经包含了合适的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

这个程序只是在载入新的要发送的数据之前，通过检测 UDREn 标志等待发送缓冲器为空。如果使用了数据寄存器空中断，则数据写入缓冲器的操作在中断程序中进行。

发送 9 位数据位的帧

如果发送 9 位数据的数据帧 (UCSZn = 7)，应先将数据的第 9 位写入寄存器 UCSRnB 的 TXB8，然后再将低 8 位数据写入发送数据寄存器 UDRn。以下程序给出发送 9 位数据的数据帧例子。在汇编代码中要发送的数据存放在 R17:R16 寄存器中。

汇编代码例程 (1)(2)

```
USART_Transmit:
  ; 等待发送缓冲器为空
  sbis  UCSRnA,UDREn
  rjmp  USART_Transmit
  ; 将第 9 位从 r17 中复制到 TXB8
  cbi   UCSRnB,TXB8
  sbrc  r17,0
  sbi   UCSRnB,TXB8
  ; 将低 8 位数据放入缓冲器，发送数据
  out   UDRn,r16
  ret
```

C 代码例程 (1)(2)

```
void USART_Transmit( unsigned int data )
{
  /* 等待发送缓冲器为空*/
  while ( !( UCSRnA & (1<<UDREn))) )
        ;
  /* 将第 9 位复制到 TXB8**/
  UCSRnB &= ~(1<<TXB8);
  if ( data & 0x0100 )
    UCSRnB |= (1<<TXB8);
  /* 将数据放入缓冲器，发送数据 */
  UDRn = data;
}
```

Notes:
1. 这些函数均为通用函数。如果 UCSRnB 的内容在应用中是固定的，函数可以进一步优化。例如，初始化后只使用 UCSRnB 寄存器的 TXB8 位。
2. 本代码假定已经包含了合适的头文件
   当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

第 9 位数据在多机通信中用于表示地址帧，在同步通信中可以用于协议处理。

传送标志位与中断

USART 发送器有两个标志位:USART 数据寄存器空标志 UDREn 及传输结束标志 TXCn，两个标志位都可以引发中断。

数据寄存器空 UDREn 标志位表示发送缓冲器是否可以接受新的数据。该位在发送缓冲器为空时置 "1" ；当发送缓冲器包含需要发送的数据时清零。为与将来的器件兼容，写 UCSRnA 寄存器时该位要写 "0"。

当 UCSRnB 寄存器的数据寄存器空中断使能位 UDRIEn 为 "1" 时，只要 UDREn 被置位 (全局中断使能)，就将产生 USART 数据寄存器空中断请求。对寄存器 UDRn 执行写操作将清零 UDREn。采用中断方式的传输数据时，在数据寄存器空中断服务程序中必须写一个新的数据到 UDRn 以清零 UDREn ；或者是禁止数据寄存器空中断。否则一旦该中断程序结束，新的中断将再次产生。

当整个数据帧移出发送移位寄存器，同时发送缓冲器中又没有新的数据时，发送结束 TXCn 标志位置位。TXCn 标志位在传送结束中断执行时自动清零，也可在该位写 "1" 来清零。 TXCn 标志位对于采用如 RS-485 标准的半双工通信接口十分有用。在这些应用里，一旦传送完毕，应用程序必须释放通信总线并进入接收状态。

当 UCSRB 的发送结束中断使能位 TXCIEn 与全局中断使能位均置 "1" 时，随着 TXCn 标志位的置位， USART 发送结束中断将被执行。一旦进入中断服务程序， TXCn 标志位即被自动清零，中断处理程序不必执行 TXCn 清零操作。

奇偶校验产生电路

奇偶校验产生电路为串行数据帧生成相应的校验位。校验位使能 (UPMn1 = 1) 时，发送控制逻辑电路会在数据的最后一位与第一个停止位之间插入奇偶校验位。

禁用发送器

TXEN 清零后，只有等到所有的数据发送完成后发送器才能够真正禁用，即发送移位寄存器与发送缓冲寄存器中没有要传送的数据。发送器禁用后， TxDn 引脚恢复其通用 I/O 功能。

数据接收—USART 接收器

置位 UCSRB 寄存器的接收允许位 (RXENn) 即可启动 USART 接收器。接收器使能后 RxDn的普通引脚功能被USART功能所取代，成为接收器的串行输入口。进行数据接收之前首先要设置好波特率、操作模式及帧格式。如果使用同步操作，XCKn 引脚上的时钟被用作传输时钟。

以 5 到 8 个数据位的方式接收帧

一旦接收器检测到一个有效的起始位，便开始接收数据。起始位后的每一位数据都以设定的波特率或 XCKn 时钟进行接收，直到收到一帧数据的第一个停止位。接收到的数据被送入接收移位寄存器。第二个停止位会被接收器忽略。接收到第一个停止位后，接收移位寄存器就包含了一个完整的数据帧。这时移位寄存器中的内容将被转移到接收缓冲器中。通过读取 UDR 就可以获得接收缓冲器的内容。

以下程序给出了对 RXCn 标志采用轮询方式接收数据的例子。当数据帧少于 8 位时，从 UDRn 读取的相应的高几位为 0。当然，执行本段代码之前首先要初始化 USART。

汇编代码例程 (1)

```
USART_Receive:
  ; 等待接收数据
  sbis  UCSRnA, RXCn
  rjmp  USART_Receive
  ; 从缓冲器中获取并返回数据
  in    r16, UDRn
  ret
```

C 代码例程 (1)

```
unsigned char USART_Receive( void )
{
  /* 等待接收数据*/
  while ( !(UCSRnA & (1<<RXCn)) )
        ;
  /* 从缓冲器中获取并返回数据*/
  return UDRn;
}
```

Note: 1. 本代码假定已经包含了相应的头文件。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

在读缓冲器并返回之前，函数通过检查 RXCn 标志来等待数据送入接收缓冲器。

以 9 个数据位的方式接收帧

如果设定了 9 位数据的数据帧 (UCSZ=7)，在从 UDR 读取低 8 位之前必须首先读取寄存器 UCSRnB 的 RXB8n 以获得第 9 位数据。这个规则同样适用于状态标志位 FEn、DORn 及 UPEn。状态通过读取 UCSRnA 获得，数据通过 UDRn 获得。读取 UDRn 存储单元会改变接收缓冲器 FIFO 的状态，进而改变同样存储在 FIFO 中的 TXB8n、FEn、DORn 与 UPEn 位。

接下来的代码示例展示了一个简单的USART接收函数，说明如何处理9位数据及状态位。

汇编代码例程 (1)

```
USART_Receive:
  ; 等待接收数据
  sbis  UCSRnA, RXCn
  rjmp  USART_Receive
  ; 从缓冲器中获得状态、第 9 位及数据
  in    r18, UCSRnA
  in    r17, UCSRnB
  in    r16, UDRn
  ; 如果出错，返回 -1
  andi  r18,(1<<FEn)|(1<<DORn)|(1<<UPEn)
  breq  USART_ReceiveNoError
  ldi   r17, HIGH(-1)
  ldi   r16, LOW(-1)
USART_ReceiveNoError:
  ;过滤第 9 位数据，然后返回
  lsr   r17
  andi  r17, 0x01
  ret
```

C 代码例程 (1)

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* 等待接收数据*/
  while ( !(UCSRnA & (1<<RXCn)) )
        ;
  /* 从缓冲器中获得状态、第 9 位及数据*/

  status = UCSRnA;
  resh = UCSRnB;
  resl = UDRn;
  /* 如果出错，返回 -1*/
  if ( status & (1<<FEn)|(1<<DORn)|(1<<UPEn) )
    return -1;
  /* 过滤第 9 位数据，然后返回*/
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. 本代码假定已经包含了相应的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

上述例子在进行任何计算之前将所有的 I/O 寄存器的内容读到寄存器文件中。这种方法优化了对接收缓冲器的利用。它尽可能早的释放了缓冲器以接收新的数据。

接收结束标志及中断

USART 接收器有一个标志用来标明接收器的状态。

接收结束标志 (RXCn) 用来说明接收缓冲器中是否有未读出的数据。当接收缓冲器中有未读出的数据时，此位为 1，当接收缓冲器空时为 0(即不包含未读出的数据)。如果接收器被禁止 (RXENn = 0)，接收缓冲器会被刷新，从而使 RXCn 清零。

置位 UCSRB 的接收结束中断使能位 ((RXCIEn) 后，只要 RXCn 标志置位 (全局中断使能) 就会产生 USART 接收结束中断。使用中断方式进行数据接收时，数据接收结束中断服务程序程序必须从 UDRn 读取数据以清除 RXCn 标志，否则只要中断处理程序一结束，新的中断就会产生。

接收器错误标志

USART 接收器有三个错误标志：：帧错误 (FEn)、数据溢出 (DORn) 及奇偶校验错 (UPEn)。它们都位于寄存器 UCSRnA。错误标志与数据帧一起保存在接收缓冲器中。由于读取 UDRn 会改变缓冲器，UCSRnA 的内容必须在读接收缓冲器 (UDRn) 之前读入。错误标志的另一个同一性是它们都不能通过软件写操作来修改。但是为了保证与将来产品的兼容性，对执行写操作是必须对这些错误标志所在的位置写 "0“。所有的错误标志都不能产生中断。

帧错误标志 (FEn) 表明了存储在接收缓冲器中的下一个可读帧的第一个停止位的状态。停止位正确 (为 1) 则 FEn 标志为 0，否则 FE 标志为 1。这个标志可用来检测同步丢失、传输中断，也可用于协议处理。UCSRnC 中 USBSn 位的设置不影响 FE 标志位，因为除了第一位，接收器忽略所有其他的停止位。为了与以后的器件相兼容，写 UCSRnA 时这一位必须置 0。

数据溢出标志 (DORn) 表明由于接收缓冲器满造成了数据丢失。当接收缓冲器满 (包含了两个字符)，接收移位寄存器又有数据，若此时检测到一个新的起始位，数据溢出就产生了。 DORn 标志位置位即表明在最近一次读取 UDRn 和下一次读取 UDRn 之间丢失了一个或更多的数据帧。为了与以后的器件相兼容，写 UCSRnA 时这一位必须置 0。当数据帧成功地从移位寄存器转入接收缓冲器后， DORn 标志被清零。

奇偶校验错标志 (UPEn) 指出接收缓冲器中的下一帧数据在接收时有奇偶错误。如果不使能奇偶校验，那么 UPEn 位应清零。为了与以后的器件相兼容，写 UCSRnA 时这一位必须置 0。细节请参照 P156” 校验位的计算 ” 与 P163” 奇偶校验器 ” 。

奇偶校验器

奇偶校验模式位 UPMn1 置位将启动奇偶校验器。校验的模式 (偶校验还是奇校验) 由 UPMn0 确定。奇偶校验使能后，校验器将计算输入数据的奇偶并把结果与数据帧的奇偶位进行比较。校验结果将与数据和停止位一起存储在接收缓冲器中。这样就可以通过读取奇偶校验错误标志位 (UPEn) 来检查接收的帧中是否有奇偶错误。

如果下一个从接收缓冲器中读出的数据有奇偶错误，并且奇偶校验使能 (UPMn1 = 1)，则 UPEn 置位。直到接收缓冲器 (UDRn) 被读取，这一位一直有效。

禁止接收器

与发送器对比，禁用接收器即刻起作用。正在接收的数据将丢失。禁用接收器 (RXENn 清零) 后，接收器将不再占用 RxDn 引脚；接收缓冲器 FIFO 也会被刷新。缓冲器中的数据将丢失。

刷新接收缓冲器

禁用接收器时缓冲器 FIFO 被刷新，缓冲器被清空。导致未读出的数据丢失。如果由于出错而必须在正常操作下刷新缓冲器，则需要一直读取 UDRn 直到 RXCn 标志清零。下面的代码给出如何刷新接收缓冲器。

汇编代码例程 (1)

```
USART_Flush:
  sbis  UCSRnA, RXCn
  ret
  in    r16, UDRn
  rjmp  USART_Flush
```

C 代码例程 (1)

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSRnA & (1<<RXCn) ) dummy = UDRn;
}
```

Note: 1. 本代码假定已经包含了相应的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

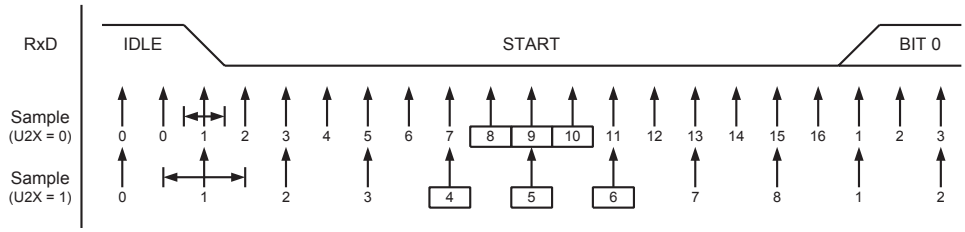
异步数据接收

USART 有一个时钟恢复单元和数据恢复单元用来处理异步数据接收。时钟恢复逻辑用于同步从 RxDn 引脚输入的异步串行数据和内部的波特率时钟。数据恢复逻辑采集数据，并通过一个低通滤波器过滤所输入的每一位数据，从而提高接收器的抗干扰性能。异步接收的工作范围依赖于内部波特率时钟的精度、帧输入的速率及一帧所包含的位数。

恢复异步时钟

时钟恢复逻辑将输入的串行数据帧与内部时钟同步起来。 Figure74 展示了对输入数据帧起始位的采样过程。普通工作模式下采样率是波特率的 16 倍，倍速工作模式下则为波特率的 8 倍。水平箭头表示由于采样而造成的同步的变化。使用倍速模式 (U2Xn = 1) 时同步变化时间更长。 RxDn 线空闲 (即没有任何通讯活动) 时，采样值为 0。

Figure 74. 起始位采样



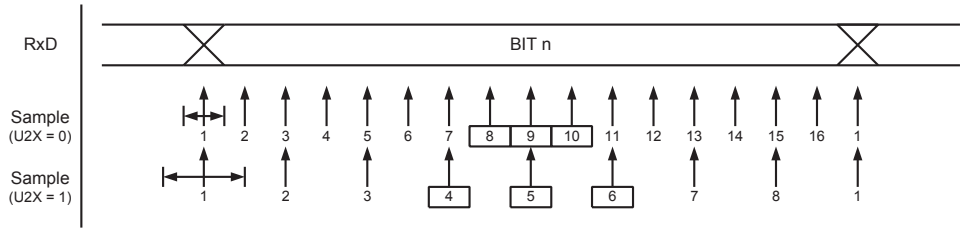
当时钟恢复电路检测到 RxD 线上一个由高 (空闲) 到低 (开始) 的电平跳变时，起始位检测序列即被启动。如图所示，我们用采样 1 表示第一个 0 采样。然后，时钟恢复逻辑用采样 8、 9、 10(普通模式)，或采样 4、 5、 6(倍速模式)，来判断是否接收到一个正确的起始位。如果这三个采样中的两个或更多个是逻辑高电平 (多数表决)，起始位会被视为毛刺噪声而被拒绝接受，接收器等待下一个由高到低的电平转换。如果检测到一个有效的起始位，时钟恢复逻辑即被同步并开始接收数据。每一个起始位都会引发同样的同步过程。

恢复异步数据

接收时钟与起始位同步之后，数据恢复工作可开始了。数据恢复单元使用一个状态机来接收每一个数据位。这个状态机在普通模式下具有 16 个状态，在倍速模式下具有 8 个状

态。 Figure75 说明了对数据位和奇偶位的采样。每个采样点都被赋予了一个数字，这个数字等于数据恢复单元当前的状态序号。

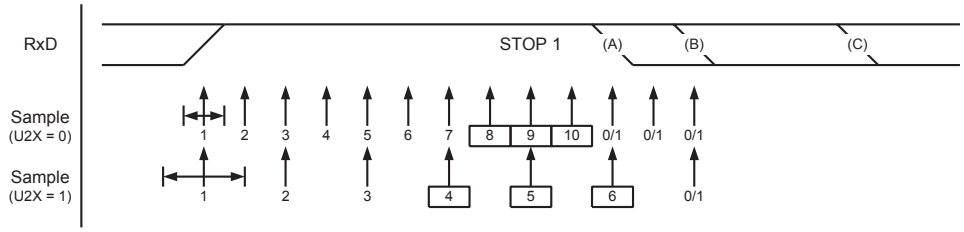
Figure 75. 数据及奇偶位的采样



确定接收到的数据位的逻辑电平的方法为多数表决法。表决对象即为三个在数据位中心获得的采样。为了强调这些采样，图中采样序号被包含在小方框中。多数表决是这样工作的：如果有 2 个或所有 3 个采样值都是高电平，那么接收位就为逻辑 1。如果 2 个或所有3个采样值都是低电平，那么接收位就被为逻辑0。对从RxD引脚输入的信号来说，多数表决的作用就象是一个低通滤波。数据恢复过程重复进行，直到接收到一个完整的数据帧。其中也包含了第一个停止位。接收器将忽略其他的停止位。

Figure76 说明了停止位的采样，以及下一帧信号起始位最早可能出现的情况。

Figure 76. 停止位及下一个起始位采样



多数表决对停止位同样有效。若停止位为逻辑 0，那么帧错误标志 FEn 置位。

如果电平再一次出现了从高到低的跳变，说明紧接着上一个数据帧来了新的数据帧。在普通模式中，第一个低电平的采样点可以发生在 Figure76 的 A 点。在倍速工作模式下第一个低电平采样点必须延迟到 B 点， C 点则为完整停止位的结束位置。对起始位的及早检测将影响接收器的工作范围。

异步工作范围

接收器的工作范围取决于接收到的数据速率及内部波特率之间的不匹配程度。如果发送器以过快或过慢的比特率传输数据帧，或者接收器内部产生的波特率没有相同的频率 (见 Table74)，那么接收器就无法与起始位同步。

下面的公式可用来计算数据输入速率与内部接收器波特率的比值。

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D 字符长度及奇偶位长度的总和 (D = 5 到 10 位)

S 每一位的采样数。普通模式下 S = 16，倍速模式下 S = 8

S_F 用于多数表决的第一个采样序号。普通模式下 S_F = 8 ，倍速模式下 S_F = 4

S_M 用于多数表决的中间采样序号。普通模式下 S_M = 9 ，倍速模式下 S_M = 5

R_{slow} 是可接受的、最慢的数据输入速率与接收器波特率的比值； R_{fast} 是可接受的、最快的数据输入速率与接收器波特率的比值。

Table74 与 Table75 列出了容许的最大接收器波特率误差。需要注意的是，普通模式下波特率允许有更大的变化范围。

Table 74. 普通模式下推荐的最大接收器波特率误差范围 (U2Xn = 0)

D # (数据 + 校验位)	R_{slow} (%)	R_{fast} (%)	最大的总误差 (%)	推荐的最大接收器误差 (%)
5	93.20	106.67	+6.67/-6.8	± 3.0
6	94.12	105.79	+5.79/-5.88	± 2.5
7	94.81	105.11	+5.11/-5.19	± 2.0
8	95.36	104.58	+4.58/-4.54	± 2.0
9	95.81	104.14	+4.14/-4.19	± 1.5
10	96.17	103.78	+3.78/-3.83	± 1.5

Table 75. 倍速率模式下推荐的最大接收器波特率误差范围 (U2Xn = 1)

D # (数据 + 校验位)	R_{slow} (%)	R_{fast} (%)	最大的总误差 (%)	推荐的最大接收器误差 (%)
5	94.12	105.66	+5.66/-5.88	± 2.5
6	94.92	104.92	+4.92/-5.08	± 2.0
7	95.52	104,35	+4.35/-4.48	± 1.5
8	96.00	103.90	+3.90/-4.00	± 1.5
9	96.39	103.53	+3.53/-3.61	± 1.5
10	96.70	103.23	+3.23/-3.30	± 1.0

上述推荐的最大接收波特率误差是在假定接收器和发送器对最大总误差具有同等贡献的前提下得出的。

产生接收器波特率误差的原因有两个。首先，接收器系统时钟 (XTAL) 的稳定性与电压范围及工作温度有关。使用晶振来产生系统时钟时一般不会有此问题，但对于谐振器而言，根据谐振器不同的误差容限，系统时钟可能有超过 2% 的偏差。第二个误差的原因就好控制多了。波特率发生器不一定能够通过对系统时钟的分频得到恰好的波特率。此时可以调整 UBRRn 值，使得误差低至可以接受。

多处理器通信模式

置位 UCSRnA 的多处理器通信模式位 (MPCMn) 可以对 USART 接收器接收到的数据帧进行过滤。那些没有地址信息的帧将被忽略，也不会存入接收缓冲器。在一个多处理器系统中，处理器通过同样的串行总线进行通信，这种过滤有效的减少了需要 CPU 处理的数据帧的数量。MPCMn 位的设置不影响发送器的工作，但在使用多处理器通信模式的系统中，它的使用方法会有所不同。

如果接收器所接收的数据帧长度为 5 到 8 位，那么第一个停止位表示这一帧包含的是数据还是地址信息。如果接收器所接收的数据帧长度为 9 位，那么由第 9 位 (RXB8n) 来确定是数据还是地址信息。如果确定帧类型的位 (第一个停止位或第 9 个数据位) 为 1，那么这是地址帧，否则为数据帧。

在多处理器通信模式下，多个从处理器可以从一个主处理器接收数据。首先要通过解码地址帧来确定所寻址的是哪一个处理器。如果寻址到某一个处理器，它将正常接收后续的数据，而其他的从处理器会忽略这些帧直到接收到另一个地址帧。

使用 MPCMn

对于作为主机的处理器来说，它可以使用 9 位数据帧格式 (UCSZn = 7)。如果传输的是地址帧 (TXB8n = 1) 就将第 9 位 (TXB8n) 置 1，如果是数据帧 (TXBn = 0) 就将它清零。在这种帧格式下，从处理器必须工作于 9 位数据帧格式。

下面即为在多处理器通信模式下进行数据交换的步骤：

1. 所有的从机都工作于多处理器通信模式 (UCSRnA 寄存器的 MPCMn 置位)。
2. 主机发送地址帧后，所有的从机都会接收并读取此帧。从机 UCSRnA 寄存器的 RXCn 正常置位。
3. 每一个从机都会读取 UDRn 寄存器的内容以确定自己是否被选中。如果选中，就清零 UCSRnA 的 MPCMn 位，否则它将等待下一个地址字节的到来，并保持 MPCMn 为 1。
4. 被寻址的从机将接收所有的数据帧，直到收到一个新的地址帧。而那些保持 MPCMn 位为 1 的从机将忽略这些数据。
5. 被寻址的处理器接收到最后一个数据帧后，它将置位 MPCMn，并等待主机发送下一个地址帧。然后第 2 步之后的步骤重复进行。

使用 5 至 8 比特的帧格式是可以的，但是不实际，因为接收器必须在使用 n 和 n+1 帧格式之间进行切换。由于接收器和发送器使用相同的字符长度设置，这种设置使得全双工操作变得很困难。如果使用 5 至 8 比特的帧格式，发送器应该设置两个停止位 (USBSn = 1)，其中的第一个停止位被用于判断帧类型。

不要使用读 - 修改 - 写指令 (SBI 和 CBI) 来操作 MPCMn 位。MPCMn 和 TXCn 标志使用相同的 I/O 单元，使用 SBI 或 CBI 指令可能会不小心将它清零。

USART 寄存器描述

USART I/O 数据寄存器—UDRn

位	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (读)
	TXB[7:0]								UDRn (写)
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

USART 发送数据缓冲寄存器和 USART 接收数据缓冲寄存器共享相同的 I/O 地址，称为 USART 数据寄存器或 UDRn。将数据写入 UDRn 时实际操作的是发送数据缓冲器存器 (TXB)，读 UDRn 时实际返回的是接收数据缓冲寄存器 (RXB) 的内容。

在 5、6、7 比特字长模式下，未使用的高位被发送器忽略，而接收器则将它们设置为 0。

只有当 UCSRnA 寄存器的 UDREn 标志置位后才可以对发送缓冲器进行写操作。如果 UDREn 没有置位，那么写入 UDRn 的数据会被 USART 发送器忽略。当数据写入发送缓冲

器后，若发送器使能，且移位寄存器为空，发送器将把数据加载到发送移位寄存器。然后数据串行地从 TxDn 引脚输出。

接收缓冲器包括一个两级 FIFO，一旦接收缓冲器被寻址 FIFO 就会改变它的状态。因此不要对这一存储单元使用读 - 修改 - 写指令 (SBI 和 CBI)。使用位查询指令 (SBIC 和 SBIS) 时也要小心，因为这也有可能改变 FIFO 的状态。

USART 控制和状态寄存器 A— UCSRnA

位	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
读 / 写	R	R/W	R	R	R	R	R/W	R/W	
初始值	0	0	1	0	0	0	0	0	

- **位 7 – RXCn: USART 接收结束**

接收缓冲器中有未读出的数据时 RXCn 置位，否则清零。接收器禁止时，接收缓冲器被刷新，导致 RXCn 清零。 RXCn 标志可用来产生接收结束中断 (见对 RXCIEn 位的描述)。

- **位 6 – TXCn: USART 发送结束**

发送移位缓冲器中的数据被送出，且当发送缓冲器 (UDRn) 为空时 TXCn 置位。执行发送结束中断时 TXCn 标志自动清零，也可以通过写 1 进行清除操作。TXCn 标志可用来产生发送结束中断 (见对 TXCIEn 位的描述)。

- **位 5 – UDREn: USART 数据寄存器空**

UDREn 标志指出发送缓冲器 (UDRn) 是否准备好接收新数据。UDREn 为 1 说明缓冲器为空，已准备好进行数据接收。 UDREn 标志可用来产生数据寄存器空中断 (见对 UDRIEn 位的描述)。

复位后 UDREn 置位，表明发送器已经就绪。

- **位 4 – FEn: 帧错误**

如果接收缓冲器接收到的下一个字符有帧错误，即接收缓冲器中的下一个字符的第一个停止位为 0，那么 FEn 置位。这一位一直有效直到接收缓冲器 (UDRn) 被读取。当接收到的停止位为 1 时， FEn 标志为 "0”。对 UCSRnA 进行写入时，这一位要写 "0”。

- **位 3 – DORn: 数据溢出**

数据溢出时 DORn 置位。当接收缓冲器满 (包含了两个数据) ,接收移位寄存器又有数据，若此时检测到一个新的起始位，数据溢出就产生了。这一位一直有效直到接收缓冲器 (UDRn) 被读取。对 UCSRnA 进行写入时，这一位要写 "0”。

- **位 2 – UPEn: USART 奇偶校验错误**

当奇偶校验使能 (UPMn1 = 1) ,且接收缓冲器中所接收到的下一个字符有奇偶校验错误时 UPEn 置位。这一位一直有效直到接收缓冲器 (UDRn) 被读取。对 UCSRnA 进行写入时，这一位要写 "0”。

- **位 1 – U2Xn: 倍速发送**

这一位仅对异步操作有影响。使用同步操作时将此位清零。

此位置 1 可将波特率分频因子从 16 降到 8 ,从而有效的将异步通信模式的传输速率加倍。

- **位 0 – MPCMn: 多处理器通信模式**

设置此位将启动多处理器通信模式。MPCMn 置位后，USART 接收器接收到的那些不包含地址信息的输入帧都将被忽略。发送器不受 MPCMn 设置的影响。详细信息请参考 P168” 多处理器通信模式 ” 。

USART 控制和状态寄存器 B— UCSRnB

位	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – RXCIEn: 接收结束中断使能**

置位后使能 RXCn 中断。当 RXCIEn 为 1 ,SREG 寄存器的全局中断标志位置位 ,UCSRnA 寄存器的 RXCn 亦为 1 时可以产生 USART 接收结束中断。

- **位 6 – TXCIEn: 发送结束中断使能**

置位后使能 TXCn 中断。当 TXCIEn 为 1 ,SREG 寄存器的全局中断标志位置位 ,UCSRnA 寄存器的 TXCn 亦为 1 时可以产生 USART 发送结束中断。

- **位 5 – UDRIEn: USART 数据寄存器空中断使能**

置位后使能 UDREn 中断。当 UDRIEn 为 1，SREG 寄存器的全局中断标志位置位，UCSRnA 寄存器的 UDREn 亦为 1 时可以产生 USART 数据寄存器空中断。

- **位 4 – RXENn: 接收使能**

置位后将启动 USART 接收器。 RxDn 引脚的通用端口功能被 USART 功能所取代。禁止接收器将刷新接收缓冲器，并使 FEn、 DORn 及 UPEn 标志无效。

- **位 3 – TXENn: 发送使能**

置位后将启动将启动 USART 发送器。TxDn 引脚的通用端口功能被 USART 功能所取代。TXENn 清零后，只有等到所有的数据发送完成后发送器才能够真正禁止，即发送移位寄存器与发送缓冲寄存器中没有要传送的数据。发送器禁止后， TxDn 引脚恢复其通用 I/O 功能。

- **位 2 – UCSZn2: 字符长度**

UCSZn2 与 UCSRnC 寄存器的 UCSZn1:0 结合在一起可以设置数据帧所包含的数据位数 (字符长度)。

- **位 1 – RXB8n: 接收数据位 8**

对 9 位串行帧进行操作时，RXB8n 是第 9 个数据位。读取 UDRn 包含的低位数据之前首先要读取 RXB8n。

- **位 0 – TXB8n: 发送数据位 8**

对 9 位串行帧进行操作时 ,TXB8n 是第 9 个数据位。写 UDRn 之前首先要对它进行写操作。

USART 控制和状态寄存器 nC— UCSRnC

位	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	1	1	0	

- **位 7:6 – UMSELn1:0 USART 模式选择**

通过这两位可以选择 USARTn 的工作模式，请见 Table76。 .

Table 76. UMSELn 位设置

UMSELn1	UMSELn0	模式
0	0	异步操作
0	1	同步操作
1	0	(保留)
1	1	SPI 主机 (MSPIM)(1)

Note: 1. SPI 主机模式 (MSPIM) 的完整描述请见 P177”SPI 模式下的 USART”

- **位 5:4 – UPMn1:0: 奇偶校验模式**

这几位设置奇偶校验的模式并使能奇偶校验。如果使能了奇偶校验，那么在发送数据，发送器都会自动产生并发送奇偶校验位。对每一个接收到的数据，接收器都会产生一奇偶值，并与 UPMn 所设置的值进行比较。如果不匹配，那么就将 UCSRnA 中的 UPEn 置位。

Table 77. UPMn 位设置

UPMn1	UPMn0	校验模式
0	0	禁止
0	1	保留
1	0	偶校验
1	1	奇校验

- **位 3 – USBSn: 停止位选择**

通过这一位可以设置停止位的位数。接收器忽略这一位的设置。

Table 78. USBSn 位设置

USBSn	停止位
0	1 位
1	2 位

- **位 2:1 – UCSZn1:0: 字符长度**

UCSZn1:0 与 UCSRnB 寄存器的 UCSZn2 一起设置数据帧包含的数据位数 (字符长度)。

Table 79. UCSZn 位设置

UCSZn2	UCSZn1	UCSZn0	字符长度
0	0	0	5 位
0	0	1	6 位
0	1	0	7 位
0	1	1	8 位
1	0	0	保留
1	0	1	保留
1	1	0	保留
1	1	1	9 位

- **位 0 – UCPOLn: 时钟极性**

这一位仅用于同步工作模式。使用异步模式时将这一位清零。 UCPOLn 设置了输出数据的改变和输入数据采样，以及同步时钟 XCKn 之间的关系。

Table 80. UCPOLn 位设置

UCPOLn	发送数据的改变 (TxDn 引脚的输出)	接收数据的采样 (RxDn 引脚的输入)
0	XCKn 上升沿	XCKn 下降沿
1	XCKn 下降沿	XCKn 上升沿

USART 波特率寄存器—UBRRL 和 UBRRH

位	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- **位 15:12 – 保留位**

- 这些位是为以后的使用而保留的。为了与以后的器件兼容，写 UBRRnH 时将这些位清零。

- **位 11:0 – UBRR11:0: USART 波特率寄存器**

这个 12 位的寄存器包含了 USART 的波特率信息。其中 UBRRnH 包含了 USART 波特率高 4 位， UBRRnL 包含了低 8 位。波特率的改变将造成正在进行的数据传输受到破坏。写 UBRRnL 将立即更新波特率分频器。

波特率设置的例子

对标准晶振及谐振器频率来说，异步模式下最常用的波特率可通过 Table81 中 UBRRn 的设置来产生。表中的粗体数据表示由此产生的波特率与目标波特率的偏差不超过 0.5%。

更高的误差也是可以接受的，但发送器的抗噪性会降低，特别是需要传输大量数据时 (参看 P165" 异步工作范围 ")。误差可以通过如下公式计算：

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 81. 通用振荡器频率下设置 UBRRn 的例子

波特率 (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
最大 (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

Note: 1. UBRRn = 0，误差 = 0.0%

Table 82. 通用振荡器频率下设置 UBRRn 的例子 (续)

波特率 (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
最大 (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRRn = 0，误差 = 0.0%

Table 83. 通用振荡器频率下设置 UBRRn 的例子 (续)

波特率 (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
最大 (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRRn = 0，误差 = 0.0%

Table 84. 通用振荡器频率下设置 UBRR 的例子 (续)

波特率 (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差	UBRRn	误差
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
最大 (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRRn = 0，误差 = 0.0%

SPI 模式下的 USART

USART 可设置成与 SPI 主机兼容的工作模式。 SPI 主机模式 (MSPIM) 的主要特性是：

- **全双工 , 三线同步数据传输**
- **主机操作**
- **支持所有四种 SPI 工作模式 (模式 0, 1, 2 与 3)**
- **首先传输 LSB 或 MSB(可配置数据次序)**
- **队列操作 (双缓冲)**
- **高分辨率的波特率发生器**
- **高速工作 (fXCKmax = fCK/2)**
- **灵活的中断**

概述

将 UMSELn1:0 都置 "1" 可以使能 MSPIM 逻辑下 的 USART。在该工作模式下 SPI 主控逻辑直接控制 USART 资源。这些资源包括发送器与接收器的移位寄存器、缓冲器，波特率发生器。校验位发生器与检测器、数据与时钟恢复逻辑，及 RX 与 TX 控制逻辑禁用 。USART RX 与 TX 控制逻辑由普通 SPI 传输控制逻辑所代替。而引脚控制与中断产生逻辑在两种工作模式下是相同的。

在两种模式下 I/O 寄存器的位置是相同的。但在 MSPIM 模式下，某些控制寄存器的功能有所改变。

时钟产生

时钟产生逻辑为发送器和接收器产生基本时钟。USART MSPIM 仅支持内部时钟发生工作模式 (主机操作)。为使在 MSPIM 模式下的 USART 正确工作，DDR_XCKn 必须置位 (即作为输出)。最好在MSPIM模式下的USART使能之前(即 TXENn与 RXENn位置位) ,置位 DDR_XCKn。

MSPIM 模式的内部时钟产生与 USART 同步主机模式是相同的。波特率或 UBRRn 可用相同的公式计算，见 Table85 :

Table 85. 波特率寄存器设置计算公式

工作模式	波特率计算公式 (1)	UBRRn 计算公式
同步主机模式	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. 波特率定义为每秒的传输位速率 (bps)

BAUD 波特率 (每秒位数， bps)

$\mathbf{f_{osc}}$ 系统时钟频率

UBRRn UBRRnH 与 UBRRnL 寄存器中数值 (0-4095)

SPI 数据模式与时序

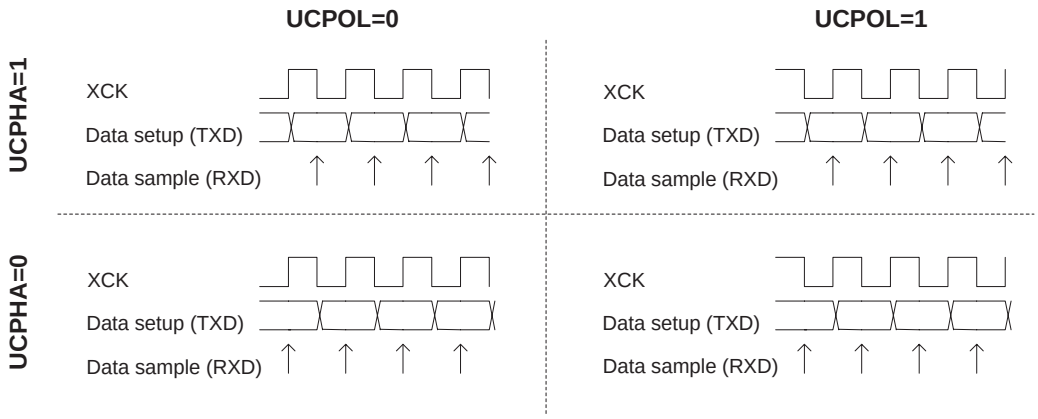
有四种 XCKn (SCK) 相位与极性的组合与串行数据有关 ,具体由 UCPHAn 与 UCPOLn 决定。数据传输的时序图请见 Figure77。数据位的移出与锁定发生在 XCKn 信号的相反边

沿，以保证有足够的时间使数据稳定。UCPOLn 与 UCPHAn 的功能总结见 Table86。改变这两位的设置将破坏正在进行的通讯。

Table 86. UCPOLn 与 UCPHAn 功能

UCPOLn	UCPHAn	SPI 模式	前沿	后沿
0	0	0	采样 (上升沿)	启动 (下降沿)
0	1	1	启动 (上升沿)	采样 (下降沿)
1	0	2	采样 (下降沿)	启动 (上升沿)
1	1	3	启动 (下降沿)	采样 (上升沿)

Figure 77. UCPHAn 与 UCPOLn 数据传输时序图



帧格式

MSPIM 的串行数据帧定义为一个字符中有 8 位数据位。MSPIM 模式下的 USART 有两种有效的帧格式：

- 8 位数据位，MSB 首先发送
- 8 位数据位， LSB 首先发送

数据帧以起始位开始，紧接着是数据字的最低位。数据字最多可以有 8 个数据位，以数据的最高位结束。最后是结束位。当一个完整的数据帧传输后，可以立即传输下一个新的数据帧，或使传输线处于空闲状态。

MSPIM 模式下的 USART 通过 UCSRnC 的 UDORDn 位设置帧格式。发送器与接收器使用相同的设置。要注意，改变这些位的设置会破坏发送器与接收器正在进行中的通信。

通过对 UDRn 写入两个数据字节可实现 16 位数据的传输。UART 传送完成中断信号将给出 16 位数据已经移出处理器的信号。

USART MSPIM 的初始化

进行通信之前首先要对 USART 进行初始化。初始化过程通常包括波特率的设定，主机模式的设定，帧结构的设定，以及根据需要使能接收器或发送器。只有发送器可独立工作。对于中断驱动的USART操作，在初始化时首先要清零全局中断标志位(全局中断被屏蔽)。

Note: 为保证即时初始化 XCKn，输出波特率寄存器 (UBRRn) 必须在发送器使能时置 "0”。与普通 USART 工作模式不同，在发送器使能后，不需要马上为 UBRRn 赋予合适的数值，而是在第一次传送开始之前对 UBRRn 赋值。如果初始化在复位之后执行，则不必在发送器使能前对 UBRRn 清零，因为在复位时 UBRRn 已经置 "0”。

重新改变 USART 的设置应该在没有数据传输的情况下进行。TXCn 标志位可以用来检验一个数据帧的发送是否已经完成，RXCn 标志位可以用来检验接收缓冲器中是否还有数据未读出。在每次发送数据之前 (在写发送数据寄存器 UDR 前)TXCn 标志位必须清零。

以下是 USART 初始化程序示例。例程采用了轮询 (中断被禁用) 的异步操作，而且帧结构是固定的。波特率作为函数参数给出。在汇编程序里波特率参数保存于寄存器对 r17:r16。

汇编代码例程 (1)

```
USART_Init:
  clr r18
  out UBRRnH,r18
  out UBRRnL,r18
  ; 将 XCKn 端口引脚设为输出，使能主机模式
  sbi XCKn_DDR, XCKn
  ; 设置 MSPI 工作模式与 SPI 数据模式 0
  ldi r18, (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn)
  out UCSRnC,r18
  ; 使能发送器与接收器
  ldi r18, (1<<RXENn)|(1<<TXENn)
  out UCSRnB,r18
  ; 设置波特率
  ; 要点：波特率的设置必须在发送器使能后
  out UBRRnH, r17
  out UBRRnL, r18
  ret
```

C 代码例程 (1)

```
void USART_Init( unsigned int baud )
{
  UBRRn = 0;
  /* 将 XCKn 端口引脚设为输出，使能主机模式 */
  XCKn_DDR |= (1<<XCKn);
  /* 设置 MSPI 工作模式与 SPI 数据模式 0*/
  UCSRnC = (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn);
  /* 使能发送器与接收器 */
  UCSRnB = (1<<RXENn)|(1<<TXENn);
  /* 设置波特率 */
  /* 要点：波特率的设置必须在发送器使能后 */
  UBRRn = baud;
}
```

Note: 1. 本代码假定已经包含了合适的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

数据发送

使用 MSPI 模式下的 USART 要求发送器使能，即 UCSRnB 寄存器中的 TXENn 位置 "1”。发送器使能时， TxDn 引脚作为发送器的串行输出，取代其普通端口功能。接收器使能是可选的，可通过对 UCSRnB 寄存器中的 RXENn 位置 "1” 来实现。接收器使能后 RxDn 引脚作为接收器的串行输输入，取代其普通端口功能。XCKn 作为传输时钟使用。

初始化后 USART 可开始传送数据。数据传送从 UDRn 写操作开始。由于传送器控制传输时钟，因此上述操作对发送与接收数据均有效。当移位寄存器准备好发送数据帧时，UDRn 的数据从传送缓冲器移入移位寄存器。

Note: 为保持输入缓冲与传送的数据字节数的同步，每个字节传送后必须对 UDRn 寄存器进行读操作。输入缓冲操作与普通的 USART 模式是一样的，即如果出现溢出，丢失的将是最后

收到的字符，而不是最先得到的数据。也就是说如果传送 4 个字节，第一是字节 1，接着是字节 2、 3、4，且在传送前未对 UDRn 读操作，则丢失的将是字节 3，而不是字节 1。

以下程序给出一个在 MSPI 模式下的 USART 对 UDRE 标志与接收结束 (RXCn) 标志采用轮询方式发送数据的例子。当然，执行本段代码之前首先要初始化 USART。在汇编代码中要发送的数据存放于 R16。

在载入新的数据之前，函数通过检测 UDREn 标志来等待传送缓冲器为空；接着函数通过检测 RXCn 标志来等待接收缓冲器获得数据；最后函数读取缓冲器的内容并返回。 .

汇编代码例程 (1)

```
USART_MSPIM_Transfer:
  ; 等待发送缓冲器为空
  sbis UCSRnA, UDREn
  rjmp USART_MSPIM_Transfer
  ; 将数据放入缓冲器，发送数据
  out UDRn,r16
  ; 等待接收数据
USART_MSPIM_Wait_RXCn:
  sbis UCSRnA, RXCn
  rjmp USART_MSPIM_Wait_RXCn
  ; 从缓冲中得到与返回接收数据
  in r16, UDRn
  ret
```

C 代码例程 (1)

```
unsigned char USART_Receive( void )
{
  /* 等待发送缓冲器为空 */
  while ( !( UCSRnA & (1<<UDREn)) );
  /* 将数据放入缓冲器，发送数据*/
  UDRn = data;
  /* 等待接收数据 */
  while ( !(UCSRnA & (1<<RXCn)) );
  /* 从缓冲中得到与返回接收数据 */
  return UDRn;
}
```

Note: 1. 本代码假定已经包含了合适的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、 “STS”、 “SBRS”、 “SBRC”、 “SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

传送器与接收器标志及中断

在 MSPIM 模式下的 USART 与普通 USART 模式下的 RXCn、TXCn 与 UDREn 及相应的中断是一致的。但在 MSPIM 模式下没有使用接收器错误状态标志 (FE、DOR 与 PE)，其返回值始终为 "0”。

禁止发送器或接收器

在 MSPIM 模式下的 USART 与普通 USART 模式下对发送器或接收器的禁用是一致的。

USART MSPIM 寄存器描述

以下部分描述 MSPIM 模式下的 USART 寄存器。

USART MSPIM I/O 数据寄存器—UDRn

在 MSPIM 模式下的 USART 与普通 USART 模式下数据寄存器 UDRn 功能与位说明是相同的，请见 P168 “USART I/O 数据寄存器 – UDRn” 。

USART MSPIM 控制和状态寄存器 A—UCSRnA

位	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	-	-	-	-	-	UCSRnA
读 / 写	R/W	R/W	R/W	R	R	R	R	R	
初始值	0	0	0	0	0	1	1	0	

- **位 7 - RXCn: USART 接收结束**

接收缓冲器中有未读出的数据时 RXCn 置位，否则清零。接收器禁止时，接收缓冲器被刷新，导致 RXCn 清零。 RXCn 标志可用来产生接收结束中断 (见对 RXCIEn 位的描述)。

- **位 6 - TXCn: USART 发送结束**

发送移位缓冲器中的数据被送出，且当发送缓冲器 (UDRn) 为空时 TXCn 置位。执行发送结束中断时 TXCn 标志自动清零，也可以通过写 "1” 进行清除操作。 TXCn 标志可用来产生发送结束中断 (见对 TXCIEn 位的描述)。

- **位 5 - UDREn: USART 数据寄存器空**

UDREn标志指出发送缓冲器(UDRn)是否准备好接收新数据。UDREn为"1”说明缓冲器为空，已准备好进行数据接收。 UDREn 标志可用来产生数据寄存器空中断 (见对 UDRIEn 位的描述)。复位后 UDREn 置位，表明发送器已经就绪。

- **位 4:0 - 在 MSPI 模式下的保留位**

在 MSPI 模式下，这几位为保留位。为与将来器件兼容，当对 UCSRnA 写入时，这几位必须置 "0”。

USART MSPIM 控制和状态寄存器 B—UCSRnB

位	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIE	RXENn	TXENn	-	-	-	UCSRnB
读 / 写	R/W	R/W	R/W	R/W	R/W	R	R	R	
初始值	0	0	0	0	0	1	1	0	

- **位 7 - RXCIEn: RX 结束中断使能**

RXCIEn 位置位使能 RXCn 标志中断。只有当 RXCIEn 位置 "1”，SREG 的全局中断标志置 "1” 且 UCSRnA 的 RXCn 位置 "1” 时，USART 接收结束中断产生。

- **位 6 - TXCIEn: TX 结束中断使能**

TXCIEn位置位使能TXCn标志中断。只有当TXCIEn位置"1”，SREG的全局中断标志置"1” 且 UCSRnA 的 TXCn 位置 "1” 时，USART 发送结束中断产生。

- **位 5 - UDRIE: USART 数据寄存器空中断使能**

UDRIE位置位使能UDREn标志中断。只有当UDRIE位置"1”，SREG的全局中断标志置"1” 且 UCSRnA 的 UDREn 位置 "1” 时，USART 数据寄存器空中断产生。

- **位 4 - RXENn: 接收器使能**

RXENn位置位使能MSPIM模式下的USART接收器。使能后接收器将替代RxDn引脚的普通端口操作。禁用接收器将刷新接收缓冲。MSPIM 模式下仅使能接收器 (即设定 RXENn=1且TXENn=0) 没有任何意义。这是因为MSPIM模式仅支持主机模式且只有发送器控制传送时钟。

- **位 3 - TXENn: 发送器使能**

TXENn位置位使能USART发送器。使能后接收器将替代TxDn引脚的普通端口操作。直到所有的传送完成即传送完发送移位寄存器与发送缓冲寄存器中的数据后，禁用发送器 (即设定 TXENn=0) 操作才生效。当发生器禁用后，发生器不再占用 TxDn 端口。

- **位 2:0 - MSPI 模式下的保留位**

在 MSPI 模式下，这几位保留。为与以后的器件兼容，在对 UCSRnB 写入时，这几位必须写入 "0”。

USART MSPIM 控制与状态寄存器 C—UCSRnC

位	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	-	-	-	UDORDn	UCPHAn	UCPOLn	UCSRnC
读 / 写	R/W	R/W	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	1	1	0	

- **位 7:6 - UMSELn1:0: USART 模式选择**

这两位为 USART 工作模式选择位，请见 Table87.P171”USART 控制和状态寄存器 n C – UCSRnC” 详细描述了普通USART操作。当两位均设为"1”时，MSPIM使能。在MSPIM模式下， UDORDn、UCPHAn 与 UCPOLn 用相同的操作设置。

Table 87. UMSELn 位设置

UMSELn1	UMSELn0	模式
0	0	异步 USART
0	1	同步 USART
1	0	(保留)
1	1	SPI 主机 (MSPIM)

- **位 5:3 - MSPI 模式下的保留位**

在 MSPI 模式下，这几位保留。为与以后的器件兼容，在对 UCSRnC 写入时，这几位必须写入 "0”。

- **位 2 - UDORDn: 数据次序**

UDORDn置"1”时，先传送数据字的LSB；否则先传送数据字的MSB，详见帧格式部分的描述。

- **位 1 - UCPHAn: 时钟相位**

UCPHAn 的设置决定数据在 XCKn 的前沿或后沿采样，详见 SPI 数据模式与时序部分。

- **位 0 - UCPOLn: 时钟极性**

UCPOLn 设置 XCKn 时钟的极性。UCPOLn 与 UCPHAn 位设定组合决定数据传送的时序，详见 SPI 数据模式与时序部分。

USART MSPIM 波特率寄存器—UBRRnL 与 UBRRnH

在 MSPI 模式下的波特率寄存器功能与位描述与普通 USART 操作的相同，详见 P172 “USART 波特率寄存器 –UBRRL 和 UBRRH” 。

AVR USART MSPIM 与 AVR SPI 的比较

MSPIM 模式下的 USART 操作在下述方面与 AVR SPI 完全兼容：

- 主机模式时序
- UCPOLn 功能与 SPI CPOL 相同
- UCPHAn 功能与 SPI CPHA 相同
- UDORDn 功能与 SPI DORD 相同

但由于 MSPIM 模式下的 USART 使用某些 USART 资源， MSPIM 模式下的 USART 与 SPI 还存在差异。除去控制寄存器位的不同及 MSPIM 模式下的 USART 仅支持主机操作模式外，两模块在下面的特性中也有不同：

- MSPIM 模式下的 USART 发送器有缓冲器， SPI 则没有
- MSPIM 模式下的 USART 接收器有附加的缓冲器
- MSPIM 模式下的 USART 没有 SPI WCOL (写冲突) 位
- MSPIM 模式下的 USART 没有 SPI 倍速模式 (SPI2X)。但是，可通过对 UBRRn 的设置达到同样的效果
- 中断时序不同
- 由于主机模式只在 MSPIM 模式下的 USART 工作，因此引脚控制不同

MSPIM 模式下的 USART 与 SPI 比较见 P182Table88 。

Table 88. MSPIM 模式下的 USART 与 SPI 引脚的比较

USART_MSPIM	SPI	建议
TxDn	MOSI	主机输出
RxDn	MISO	主机输入
XCKn	SCK	(功能相同)
(N/A)	$\overline{SS}$	MSPIM 模式下的 USART 不支持

两线串行接口

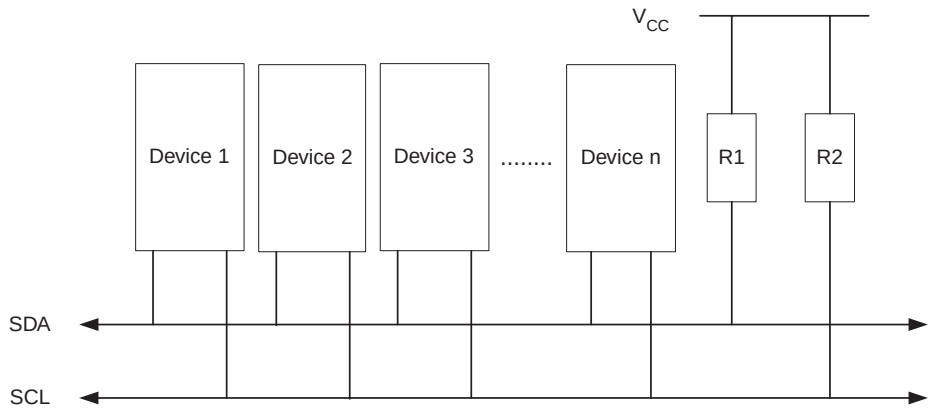
特性

- 只需两根线的、简单而功能强大、灵活的串行通信接口
- 支持主机 / 从机操作模式
- 器件可作为发送器或接收器
- 7 位地址空间，支持最大 128 个从机地址
- 支持多主机模式
- 高达 400 kHz 的数据传输率
- 斜率受限的输出驱动器
- 噪声监控电路防止总线上的毛刺
- 可编程的从机地址，支持呼叫功能
- 地址识别中断可以将 AVR 从休眠模式唤醒

两线串行接口总线定义

两线串行接口 (TWI) 是单片机应用的理想接口。采用 TWI 协议，系统设计者可以通过两根双向的总线 (一根称为时钟线 SCL，另一根称为数据线 SDA) 连接 128 个从设备。实现这种总线连接时，唯一需要增加的外部器件是每个总线上的上拉电阻。所有与总线连接的设备都有各自的设备地址。

Figure 78. TWI 总线配置



TWI 术语

以下定义在本节中将会经常出现。

Table 89. TWI 术语

术语	说明
Master	启动与终止数据传送的器件。主机同时还产生 SCL 时钟
Slave	被主机寻址的器件
Transmitter	将数据送到总线的器件
Receiver	从总线读取数据的器件

电气连接

如 Figure78 所示，总线通过上拉电阻与电源正极相联。所有 TWI 兼容的器件的总线驱动都是漏极开路或集电极开路的。这样就实现了对接口操作非常关键的线与功能。 TWI 器件输出为 "0” 时， TWI 总线会产生低电平。当所有的 TWI 器件输出为三态时，总线会输出高电平，允许上拉电阻将电压拉高。注意，为保证所有的总线操作，凡是与 TWI 总线连接的 AVR 器件必须上电。

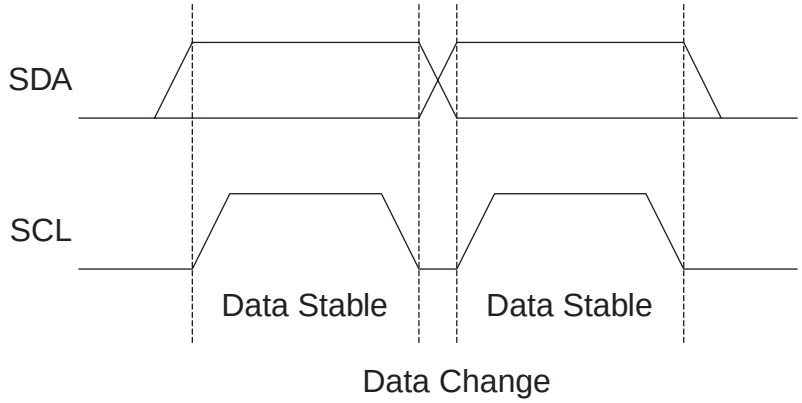
与总线连接的器件数目受如下条件限制：总线电容要低于 400 pF，而且可以用 7 位从机地址进行寻址。TWI 详细的电气特性说明请见 P280”SPI 时序特性 ” 。这儿给出了两个不同的规范，一种是总线速度低于 100 kHz，而另外一种是总线速度高达 400 kHz。

数据传输与帧格式

传输的数据位

TWI 总线上数据位的传送与时钟脉冲同步。时钟线为高时，数据线电压必须保持稳定，除非在启动与停止的状态下。

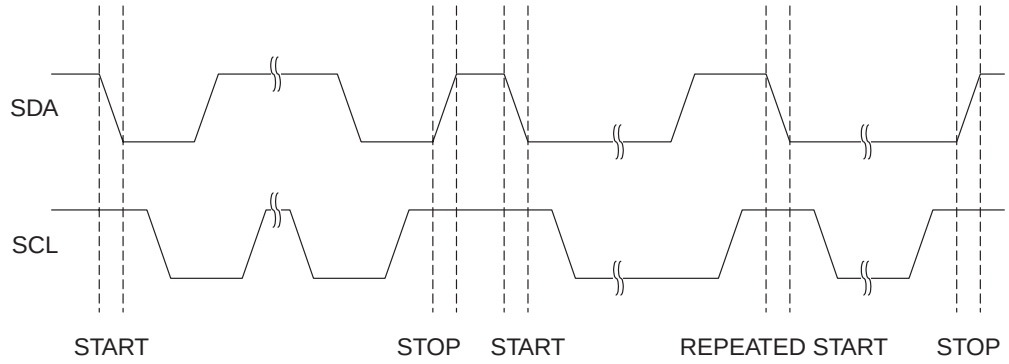
Figure 79. 数据有效性



START 与 STOP 状态

主机启动与停止数据传输。主机在总线上发出 START 信号以启动数据传输；在总线上发出 STOP 信号以停止数据传输。在 START 与 STOP 状态之间，需要假定总线忙，不允许其它主机控制总线。特例是在 START 与 STOP 状态之间发出一个新的 START 状态。这被称为 REPEATED START 状态，适用于主机在不放弃总线控制的情况下启动新的传送。在 REPEATED START 之后，直到下一个 STOP，需要假定总线处于忙的状态。这与 START 是完全一样的，因此在本手册中，如果没有特殊说明，START 与 REPEATED START 均用 START 表述。如下所示，START 与 STOP 状态是在 SCL 线为高时，通过改变 SDA 电平来实现的。

Figure 80. START、 REPEATED START 与 STOP 状态



地址包格式

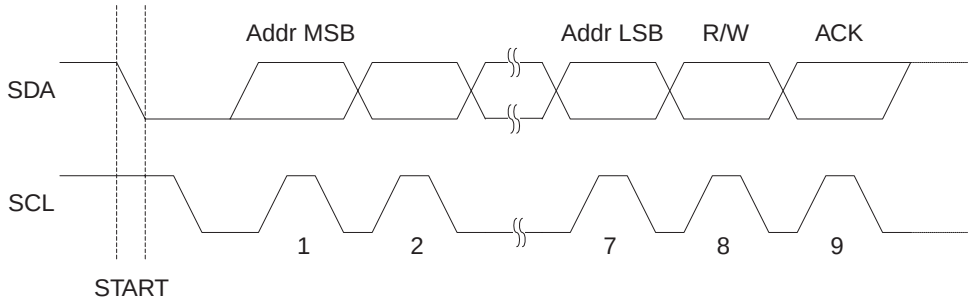
所有在 TWI 总线上传送的地址包均为 9 位，包括 7 位地址位、 1 位 READ/WRITE 控制位与 1 位应答位。如果 READ/WRITE 为 1，则执行读操作；否则执行写操作。从机被寻址后，必须在第九个 SCL (ACK) 周期通过拉低 SDA 作出应答。若该从机忙或有其它原因无法响应主机，则应该在 ACK 周期保持 SDA 为高。然后主机可以发出 STOP 状态或 REPEATED START 状态重新开始发送。地址包包括从机地址与分别称为 SLA+R 或 SLA+W 的 READ 或 WRITE 位。

地址字节的 MSB 首先被发送。从机地址由设计者自由分配，但需要保留地址 0000 000 作为广播地址。

当发送广播呼叫时，所有的从机应在 ACK 周期通过拉低 SDA 作出应答。当主机需要发送相同的信息给多个从机时可以使用广播功能。当 Write 位在广播呼叫之后发送，所有的从机通过在 ACK 周期通过拉低 SDA 作出响应。所有的从机接收到紧跟的数据包。注意在整体访问中发送 Read 位没有意义，因为如果几个从机发送不同的数据会带来总线冲突。

所有形如 1111 xxx 格式的地址都需要保留，以便将来使用。

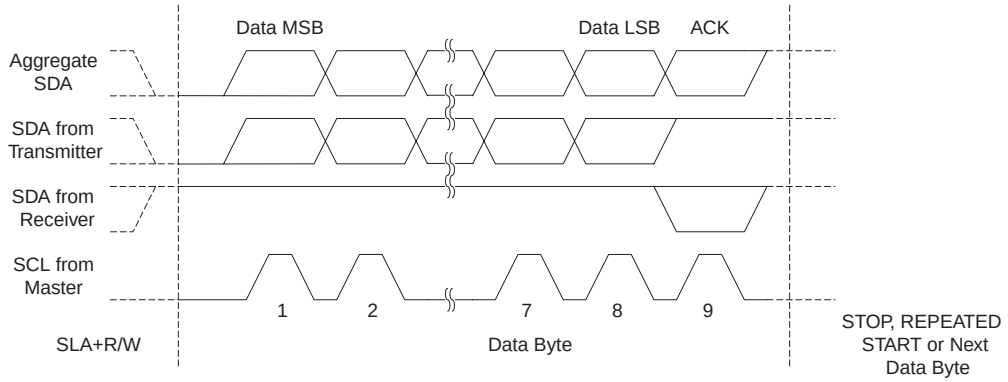
Figure 81. 地址包格式



数据包格式

所有在 TWI 总线上传送的数据包为 9 位长，包括 8 位数据位及 1 位应答位。在数据传送中，主机产生时钟及 START 与 STOP 状态，而接收器响应接收。应答是由从机在第 9 个 SCL 周期拉低 SDA 实现的。如果接收器使 SDA 为高，则发出 NACK 信号。接收器完成接收，或者由于某些原因无法接收更多的数据，应该在收到最后的字节后发出 NACK 来告知发送器。数据的 MSB 首先发送。

Figure 82. 数据包格式

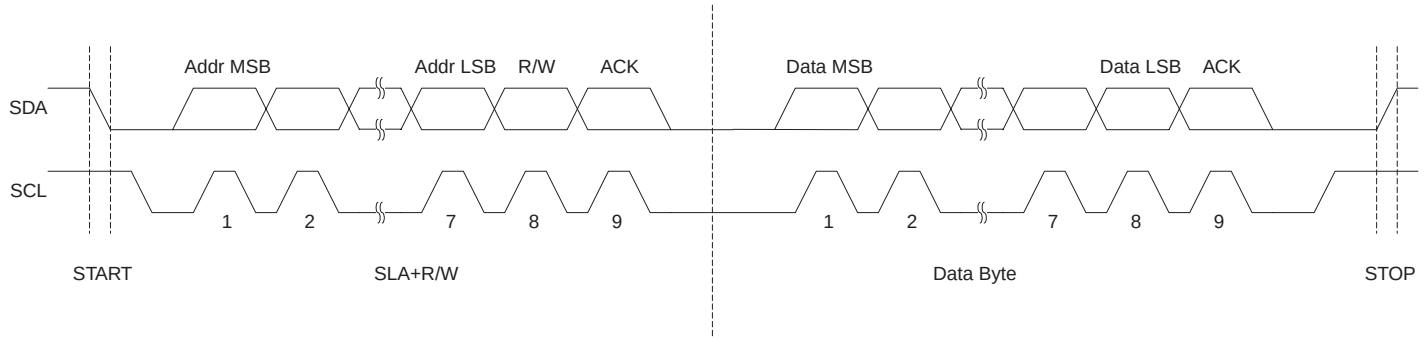


地址与数据一起发送

发送主要由 START 状态、SLA+R/W、至少一个数据包及 STOP 状态组成。只有 START 与 STOP 状态的空信息是非法的。可以利用 SCL 的线与功能来实现主机与从机的握手。从机可通过拉低 SCL 来延长 SCL 低电平的时间。当主机设定的时钟速度相对于从机太快，或从机需要额外的时间来处理数据时，这一特性是非常有用的。从机延长 SCL 低电平的时间不会影响 SCL 高电平的时间，因为 SCL 高电平时间是由主机决定的。由上述可知，通过改变 SCL 的占空比可降低 TWI 数据传送速度。

Figure83 说明了典型的数据传送。注意SLA+R/W 与STOP之间传送的字节数由应用程序的协议决定。

Figure 83. 典型的数据传送



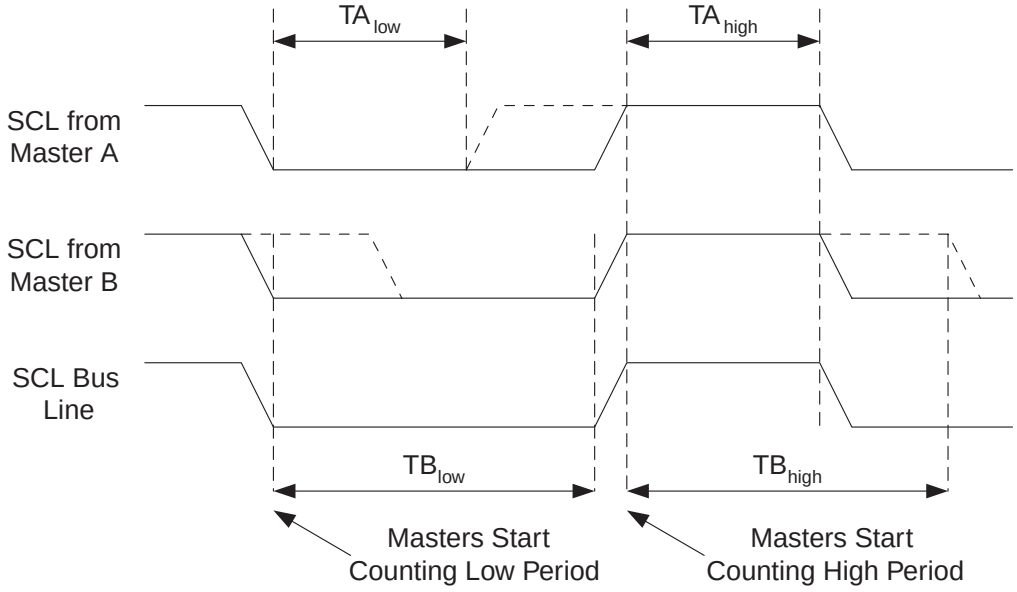
多主机总线系统，仲裁与同步

TWI 协议允许总线上由多个主机。特别要注意的是即使有多个主机同时开始发生数据，也要保证发送正常进行。多主机系统中有两个问题：

- 算法必须只能允许一个主机完成传送。当其余主机发现它们失去选择权后应停止传送。这个选择过程称为仲裁。当竞争中的主机发现其仲裁失败，应立即转换到从机模式检测是否被获得总线控制权的的主机寻址。事实上多主机同时传送时不应该让从机检测到，即不许破坏数据在总线上的传送。
- 不同的主机可能使用不同的 SCL 频率。为保证传送的一致性，必须设计一种同步主机时钟的方案。这会简化仲裁过程。

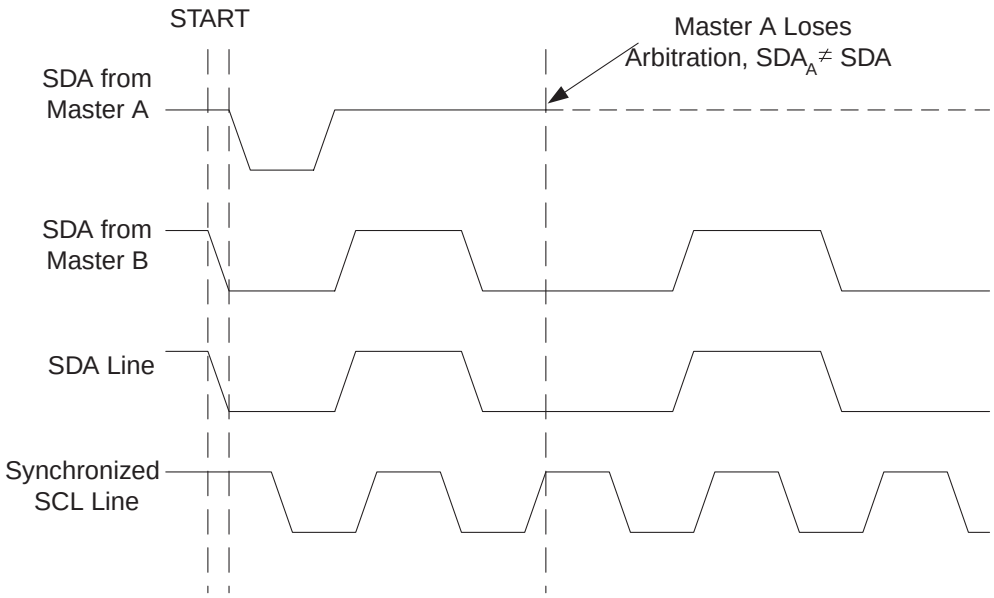
总线的线与功能用来解决上述问题。将所有的主机时钟进行与操作，会生成组合的时钟，其高电平时间等于所有主机中最短的一个；低电平时间则等于所有主机中最长的一个。所有的主机都监听 SCL，使其可以有效地计算本身高 / 低电平与组合 SCL 信号高 / 低电平的时间差异。

Figure 84. 多主机 SCL 的同步



输出数据之后所有的主机都持续监听 SDA 来实现仲裁。如果从 SDA 读回的数值与主机输出的数值不匹配，该主机即失去仲裁。要注意只有当一个主机输出高电平的 SDA，而其它主机输出为低，该主机才会失去仲裁，并立即转为从机模式，检测是否被胜出的主机寻址。失去仲裁的主机必须将 SDA 置高，但在当前的数据或地址包结束之前还可以产生时钟信号。仲裁将会持续到系统只有一个主机。这可能会占用许多比特。如果几个主机对相同的从机寻址，仲裁将会持续到数据包。

Figure 85. 两主机之间的仲裁



注意不允许在以下情况进行仲裁：

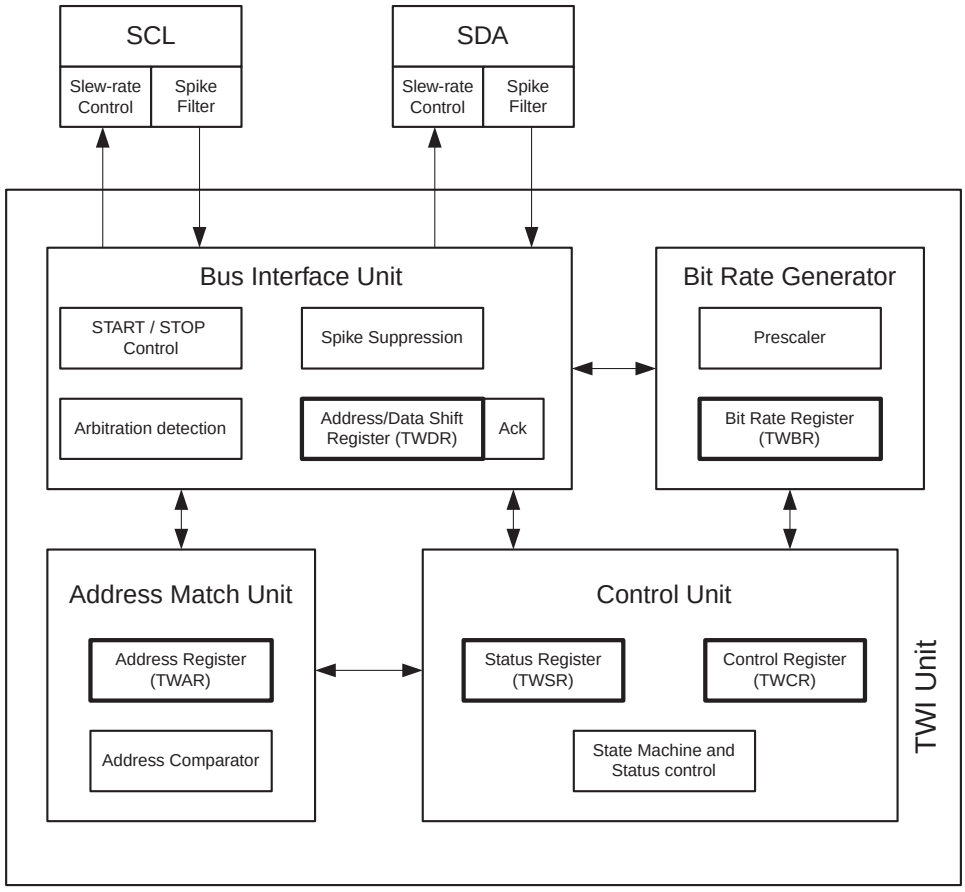
- 一个 REPEATED START 状态与一个数据位
- 一个 STOP 状态与一个数据位
- 一个 REPEATED START 状态与一个 STOP 状态

应用软件应考虑上述情况，保证不会出现这些非法仲裁状态。这意味着在多主机系统中，所有的数据传输必须由相同的 SLA+R/W 与数据包组合组成。换句话说：所有的传送必须包含相同数目的数据包，否则仲裁结果无法定义。

TWI 模块概述

TWI模块由几个子模块组成，如Figure86所示。所有位于粗线之中的寄存器可以通过AVR数据总线进行访问。

Figure 86. TWI 模块概述



SCL 与 SDA 引脚

SCL 与SDA为MCU的 TWI接口引脚。引脚的输出驱动器包含一个波形斜率限制器以满足TWI 规范。引脚的输入部分包括尖峰抑制单元以去除小于 50 ns 的毛刺。当相应的端口设置为 SCL 与 SDA 引脚时，可以使能 I/O 口内部的上拉电阻，这样可省掉外部的上拉电阻。

比特率发生器

TWI工作于主机模式时,比特率发生器控制时钟信号SCL的周期。具体由TWI状态寄存器TWSR的预分频系数以及比特率寄存器TWBR设定。当TWI工作在从机模式时,不需要对比特率或预分频进行设定,但从机的 CPU 时钟频率必须大于 TWI 时钟线 SCL 频率的 16 倍。注意,从机可能会延长 SCL 低电平的时间,从而降低 TWI 总线的平均时钟周期。SCL 的频率根据以下的公式产生:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWBR = TWI 比特率寄存器的数值
- TWPS = TWI 状态寄存器预分频的数值

Note: TWI 工作在主机模式时,TWBR 值应该不小于 10。否则主机会在 SDA 与 SCL 产生错误输出作为提示信号。问题出现于 TWI 工作在主机模式下,向从机发送 Start + SLA + R/W 的时候 (不需要真的有从机与总线连接)。

总线接口单元

该单元包括数据与地址移位寄存器 TWDR,START/STOP 控制器和总线仲裁判定硬件电路。TWDR 寄存器用于存放发送或接收的数据与地址。除了 8 位的 TWDR,总线接口单元还有一个寄存器,包含了用于发送或接收应答的 (N)ACK。这个 (N)ACK 寄存器不能由程序直接访问。当接收数据时,它可以通过 TWI 控制寄存器 TWCR 来置位或清零;在发送数据时, (N)ACK 值由 TWCR 的设置决定。

START/STOP 控制器负责产生和检测 TWI 总线上的 START、REPEATED START 与 STOP 状态。即使在 MCU 处于休眠状态时,START/STOP 控制器仍然能够检测 TWI 总线上的 START/STOP 条件,当检测到自己被 TWI 总线上的主机寻址时,将 MCU 从休眠状态唤醒。

如果 TWI 以主机模式启动了数据传输,仲裁检测电路将持续监听总线,以确定是否可以通过仲裁获得总线控制权。如果总线仲裁单元检测到自己在总线仲裁中丢失了总线控制权,则通知 TWI 控制单元执行正确的动作,并产生合适的状态码。

地址匹配单元

地址匹配单元将检测从总线上接收到的地址是否与 TWAR 寄存器中的 7 位地址相匹配。如果 TWAR 寄存器的 TWI 广播应答识别使能位 TWGCE 为 "1",从总线接收到的地址也会与广播地址进行比较。一旦地址匹配成功,控制单元将得到通知以进行正确地响应。TWI可以响应,也可以不响应主机的寻址,这取决于TWCR寄存器的设置。即使MCU处于休眠状态时,地址匹配单元仍可继续工作。一旦主机寻址到这个器件,就可以将 MCU 从休眠状态唤醒。在 TWI 由于地址匹配将 MCU 从掉电状态唤醒期间,如有其他中断发生,TWI 将放弃操作,并返回到空闲状态。如果这会引起其他问题,那么在进入掉电休眠模式之前需要确保只有 TWI 地址匹配中断被使能。

控制单元

控制单元监听 TWI 总线,并根据 TWI 控制寄存器 TWCR 的设置作出相应的响应。当 TWI 总线上产生需要应用程序干预处理的事件时,TWI 中断标志位 TWINT 置位。在下一个时钟周期, TWI 状态寄存器 TWSR 被表示这个事件的状态码字所更新。在其它时间里,TWSR 的内容为一个表示无事件发生的特殊状态字。一旦 TWINT 标志位置 "1",时钟线 SCL 即被拉低,暂停 TWI 总线上的数据传输,让用户程序处理事件。

在下列状况出现时, TWINT 标志位置位:

- 在 TWI 传送完 START/REPEATED START 信号之后
- 在 TWI 传送完 SLA+R/W 数据之后
- 在 TWI 传送完地址字节之后
- 在 TWI 总线仲裁失败之后
- 在 TWI 被主机寻址之后 (广播方式或从机地址匹配)
- 在 TWI 接收到一个数据字节之后
- 作为从机工作时, TWI 接收到 STOP 或 REPEATED START 信号之后

- 由于非法的 START 或 STOP 信号造成总线错误时

TWI 寄存器说明

TWI 比特速寄存器—TWBR

位	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..0 – TWI 比特率寄存器**

TWBR 为比特率发生器分频因子。比特率发生器是一个分频器 ,在主机模式下产生SCL时钟频率。比特率计算公式请见 P191” 比特率发生器 ” 。

TWI 控制寄存器—TWCR

位	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
读 / 写	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
初始值	0	0	0	0	0	0	0	0	

TWCR 用来控制TWI操作。它用来使能TWI ,通过施加START到总线上来启动主机访问，产生接收器应答，产生 STOP 状态，以及在写入数据到 TWDR 寄存器时控制总线的暂停等。这个寄存器还可以给出在 TWDR 无法访问期间 ,试图将数据写入到 TWDR 而引起的写入冲突信息。

- **位 7 – TWINT: TWI 中断标志**

当 TWI 完成当前工作 ,希望应用程序介入时 TWINT 置位。若 SREG 的 I 标志以及 TWCR 寄存器的 TWIE 标志也置位，则 MCU 执行 TWI 中断例程。当 TWINT 置位时， SCL 信号的低电平被延长。 TWINT 标志的清零必须通过软件写 "1” 来完成。执行中断时硬件不会自动将其改写为 "0”。要注意的是，只要这一位被清零， TWI 立即开始工作。因此，在清零 TWINT 之前一定要首先完成对地址寄存器 TWAR ,状态寄存器 TWSR ,以及数据寄存器 TWDR 的访问。

- **位 6 – TWEA: 使能 TWI 应答**

TWEA 标志控制应答脉冲的产生。若 TWEA 置位，出现如下条件时接口发出 ACK 脉冲 :

1. 器件的从机地址与主机发出的地址相符合
2. TWAR 的 TWGCE 置位时接收到广播呼叫
3. 在主机 / 从机接收模式下接收到一个字节的数据

将 TWEA 清零可以使器件暂时脱离总线。置位后器件重新恢复地址识别。

- **位 5 – TWSTA: TWI START 状态标志**

当 CPU 希望自己成为总线上的主机时需要置位 TWSTA。 TWI 硬件检测总线是否可用。若总线空闲，接口就在总线上产生 START 状态。若总线忙，接口就一直等待，直到检测到一个 STOP 状态 ,然后产生 START 以声明自己希望成为主机。发送 START 之后软件必须清零 TWSTA。

- **位 4 – TWSTO: TWI STOP 状态标志**

在主机模式下 ,如果置位 TWSTO , TWI 接口将在总线上产生 STOP 状态 ,然后 TWSTO 自动清零。在从机模式下 ,置位 TWSTO 可以使接口从错误状态恢复到未被寻址的状态。

此时总线上不会有 STOP 状态产生，但 TWI 返回一个定义好的未被寻址的从机模式且释放 SCL 与 SDA 为高阻态。

- **位 3 – TWWC: TWI 写碰撞标志**

当 TWINT 为低时写数据寄存器 TWDR 将置位 TWWC。每一次对 TWDR 的写访问都将更新此标志。

- **位 2 – TWEN: TWI 使能**

TWEN 位用于使能TWI操作与激活TWI接口。当TWEN位被写为"1”时，TWI引脚将I/O引脚切换到 SCL 与 SDA 引脚，使能波形斜率限制器与尖峰滤波器。如果该位清零， TWI 接口模块将被关闭，所有 TWI 传输将被终止。

- **位 1 – Res: 保留**

- 保留，读返回值为 ”0”。

- **位 0 – TWIE: 使能 TWI 中断**

当 SREG 的 I 以及 TWIE 置位时，只要 TWINT 为 "1”， TWI 中断就激活。

TWI 状态寄存器—TWSR

位	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	1	1	1	1	1	0	0	0	

- **位 7..3 – TWS: TWI 状态**

这 5 位用来反映 TWI 逻辑和总线的状态。不同的状态代码将会在后面的部分描述。注意从 TWSR 读出的值包括 5 位状态值与 2 位预分频值。检测状态位时设计者应屏蔽预分频位为 "0”。这使状态检测独立于预分频器设置。在无特殊声明的情况下，在手册中使用该方法。

- **位 2 – Res: 保留**

保留，读返回值为 "0”。

- **位 1..0 – TWPS: TWI 预分频位**

这两位可读 / 写，用于控制比特率预分频因子。

Table 90. TWI 比特率预分频因子

TWPS1	TWPS0	预分频值
0	0	1
0	1	4
1	0	16
1	1	64

如何计算比特率请见 P191” 比特率发生器 ” 。 TWPS1..0 值在公式中使用。

TWI 数据寄存器—TWDR

位	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	1	1	1	1	1	1	1	1	

在发送模式， TWDR 包含了要发送的字节；在接收模式， TWDR 包含了接收到的数据。当 TWI 接口没有进行移位工作 (TWINT 置位) 时这个寄存器是可写的。在第一次中断发生之前用户不能够初始化数据寄存器。只要 TWINT 置位，TWDR 的数据就是稳定的。在数据移出时，总线上的数据同时移入寄存器。 TWDR 总是包含了总线上出现的最后一个字节，除非 MCU 是从掉电或省电模式被 TWI 中断唤醒。此时 TWDR 的内容没有定义。总线仲裁失败时，主机将切换为从机，但总线上出现的数据不会丢失。 ACK 的处理由 TWI 逻辑自动管理， CPU 不能直接访问 ACK。

- **位 7..0 – TWD: TWI 数据寄存器**

根据状态的不同，其内容为要发送的下一个字节，或是接收到的数据。

TWI 从机地址寄存器—TWAR

位	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	1	1	1	1	1	1	1	0	

TWAR 的高 7 位为从机地址。工作于从机模式时，TWI 将根据这个地址进行响应。主机模式不需要此地址。在多主机系统中， TWAR 需要进行设置以便其他主机访问自己。

TWAR 的 LSB 用于识别广播地址 (0x00)。器件内有一个地址比较器。一旦接收到的地址和本机地址一致，芯片就请求中断。

- **位 7..1 – TWA: TWI 从机地址寄存器**

其值为从机地址。

- **位 0 – TWGCE: 使能 TWI 广播识别**

置位后 MCU 可以识别 TWI 总线广播。

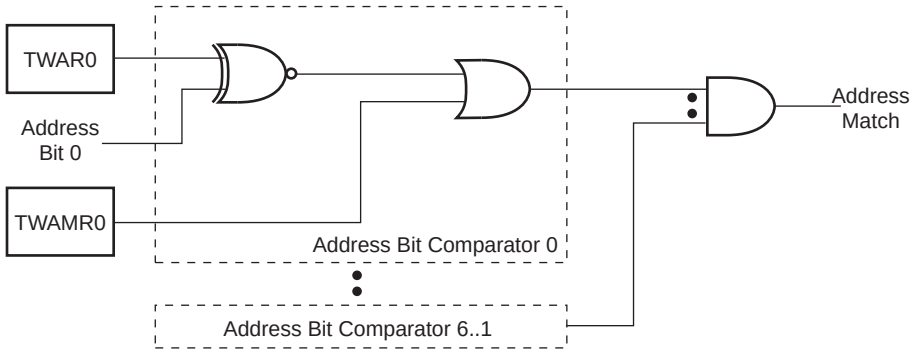
TWI 从机地址屏蔽寄存器— TWAMR

位	7	6	5	4	3	2	1	0	
	TWAM[6:0]							–	TWAMR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	
初始值	0	0	0	0	0	0	0	0	

- **位 7..1 – TWAM: TWI 地址屏蔽**

TWAMR 中装载 7 位从机地址屏蔽位。TWAMR 寄存器的每一位可禁止 TWI 地址寄存器 TWAR中相应的地址位。如果屏蔽位置"1”，则地址匹配逻辑忽略输入的地址位与TWAR相应位的比较结果。 Figure87 详细给出地址匹配逻辑。

Figure 87. TWI 地址匹配逻辑方框图



- **位 0 – Res: 保留位**

ATmega48/88/168 没有使用这些位，读返回值为 "0”。

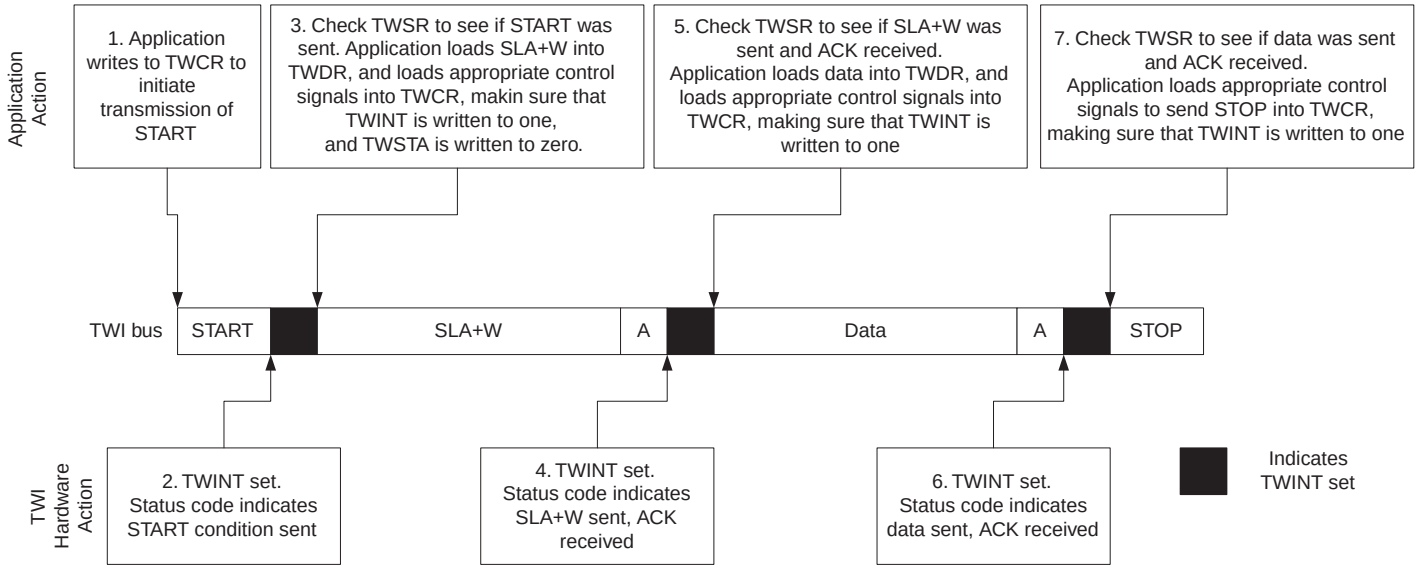
使用 TWI

AVR的 TWI接口是面向字节和基于中断的。所有的总线事件，如接收到一个字节或发送了一个 START 信号等，都会产生一个 TWI 中断。由于 TWI 接口是基于中断的，因此 TWI 接口在字节发送和接收过程中，不需要应用程序的干预。TWCR 寄存器的 TWI 中断允许 TWIE位和SREG寄存器的全局中断允许位一起决定了应用程序是否响应TWINT标志位产生的中断请求。如果 TWIE 被清零，应用程序只能采用轮询 TWINT 标志位的方法来检测 TWI 总线状态。

当 TWINT 标志位置 "1” 时，表示 TWI 接口完成了当前的操作，等待应用程序的响应。在这种情况下，TWI 状态寄存器 TWSR 包含了表明当前 TWI 总线状态的值。应用程序可以读取 TWCR 的状态码，判别此时的状态是否正确，并通过设置 TWCR 与 TWDR 寄存器，决定在下一个 TWI 总线周期 TWI 接口应该如何工作。

Figure88 给出应用程序与 TWI 接口连接的例子。该例中，主机发送一个数据字节给从机。这里只是简述，本节的后面会有更多的解释，还有简单的代码例程。

Figure 88. 典型数据传输中应用程序与 TWI 的接口



1. TWI 传输的第一步是发送 START 信号。通过对 TWCR 写入特定值，指示 TWI 硬件发送 START 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 由 START 信号启动数据传输。
2. START 信号被发送后，TWCR 寄存器的 TWINT 标志位置位，TWCR 更新为新的状态码，表示 START 信号成功发送。
3. 应用程序应检验 TWSR，确定 START 信号已成功发送。如果 TWSR 显示为其它，应用程序可以执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致，应用程序必须将 SLA+W 载入 TWDR。TWDR 可同时在地址与数据中使用。TWDR 载入 SLA+W 后，TWCR 必须写入特定值指示 TWI 硬件发送 SLA+W 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动地址包的传送。
4. 地址包发送后，TWCR 寄存器的 TWINT 标志位置位，TWDR 更新为新的状态码，表示地址包成功发送。状态代码还会反映从机是否响应包。
5. 应用程序应检验 TWSR，确定地址包已成功发送、ACK 为期望值。如果 TWSR 显示为其它，应用程序可能执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致，应用程序必须将数据包载入 TWDR。随后， TWCR 必须写入特定值指示 TWI 硬件发送 TWDR 中的数据包。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。 TWCR 寄存器中的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动数据包的传输。
6. 数据包发送后，TWCR 寄存器的 TWINT 标志位置位，TWSR 更新为新的状态码，表示数据包成功发送。状态代码还会反映从机是否响应包。
7. 应用程序应检验 TWSR，确定地址包已成功发送、ACK 为期望值。如果 TWSR 显示为其它，应用程序可能执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致， TWCR 必须写入特定值指示 TWI 硬件发送 STOP 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器中的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动 STOP 信号的传送。注意 TWINT 在 STOP 状态发送后不会置位。

尽管示例比较简单，但它包含了 TWI 数据传输过程中的所有规则。总结如下：

- 当 TWI 完成一次操作并等待反馈时， TWINT 标志置位。直到 TWINT 清零，时钟线 SCL 才会拉低。
- TWINT标志置位时，用户必须用与下一个TWI总线周期相关的值更新TWI寄存器。例如， TWDR 寄存器必须载入下一个总线周期中要发送的值。
- 当所有的 TWI 寄存器得到更新，而且其它挂起的应用程序也已经结束， TWCR 被写入数据。写 TWCR 时， TWINT 位应置位。对 TWINT 写 "1" 清除此标志。 TWI 将开始执行由 TWCR 设定的操作。

下面给出了汇编与 C 语言例程。

	汇编代码例程	C 代码例程	说明

1	`ldi r16, (1<<TWINT)|(1<<TWSTA)|`
	`(1<<TWEN)`
	`out TWCR, r16`
		`TWCR = (1<<TWINT)|(1<<TWSTA)|`
		`(1<<TWEN)`
			发出 START 信号

2	`wait1:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait1`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			等待 TWINT 置位， TWINT 置位表示 START 信号已发出

3	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, START`
	`brne ERROR`
		`if ((TWSR & 0xF8) != START)`
		`ERROR();`
			检验 TWI 状态寄存器，屏蔽预分频位，如果状态字不是 START 转出错处理

	`ldi r16, SLA_W`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = SLA_W;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			装入 SLA_W 到 TWDR 寄存器，TWINT 位清零，启动发送地址

4	`wait2:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait2`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			等待 TWINT 置位， TWINT 置位表示总线命令 SLA+W 已发出，及收到应答信号 ACK/NACK

5	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_SLA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_SLA_ACK)`
		`ERROR();`
			检验 TWI 状态寄存器，屏蔽预分频位，如果状态字不是 MT_SLA_ACK 转出错处理

	`ldi r16, DATA`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = DATA;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			装入数据到 TWDR 寄存器，TWINT 清零，启动发送数据

6	`wait3:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait3`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			等待 TWINT 置位， TWINT 置位表示总线数据 DATA 已发送，及收到应答信号 ACK/NACK

7	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_DATA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_DATA_ACK)`
		`ERROR();`
			检验 TWI 状态寄存器，屏蔽预分频器，如果状态字不是 MT_DATA_ACK 转出错处理

	`ldi r16, (1<<TWINT)|(1<<TWEN)|`
	`(1<<TWSTO)`
	`out TWCR, r16`
		`TWCR = (1<<TWINT)|(1<<TWEN)|`
		`(1<<TWSTO);`
			发送 STOP 信号

发送模式

TWI 可以工作于 4 个不同的模式 : 主机发送器 (MT)、主机接收器 (MR)、从机发送器 (ST) 及从机接收器 (SR)。同一应用程序可以使用几种模式。例如， TWI 可用 MT 模式给 TWI EEPROM 写入数据 ,用 MR 模式从 EEPROM 读取数据。如果系统中有其它主机存在 ,它们可能给 TWI 发送数据，此时就可以用 SR 模式。应用程序决定采用何种模式。

下面对每种模式进行具体说明。每种模式的状态码在详细说明数据发送的图中进行描述。这些图包含了如下的缩写：

S：START 状态

Rs：REPEATED START 状态

R：读一个比特 (SDA 为高电平)

W：写一个比特 (SDA 为低电平)

A：应答位 (SDA 为低电平)

$\overline{A}$：无应答位 (SDA 为高电平)

Data：8 位数据

P：STOP 状态

SLA：从机地址

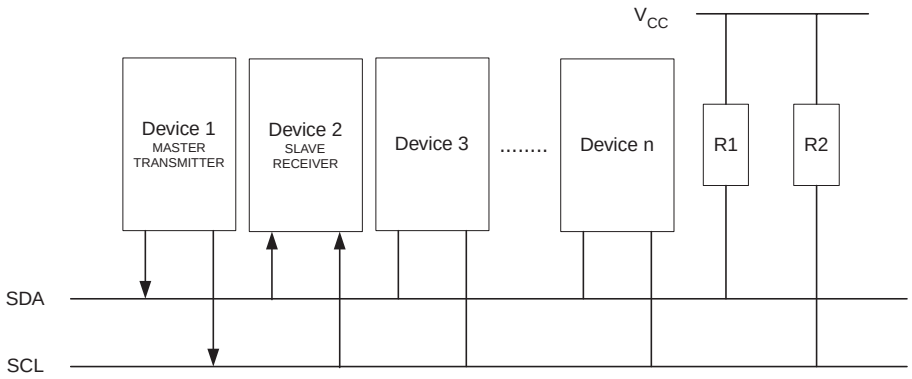
在 Figure90 到 Figure96 中，圆圈用来说明 TWINT 标志已经置位。圆圈中的数字用来表示预 TWSR 的数值，其中分频位屏蔽为 0。在这些地方应用程序必须执行合适的工作以继续 / 完成 TWI 传输。 TWI 传输被挂起，一直到 TWINT 标志被软件清零。

TWINT标志置位后 ,TWSR的状态码用来决定适当的软件操作。Table91 到Table94给出了每一个状态码所需的软件工作和后续串行传输的细节。注意在这些表中预分频位屏蔽为 0。

主机发送模式

在主机发送模式，主机可以向从机发送数据，如 Figure89 所示。为进入主机模式，必须发送 START 信号。紧接着的地址包格式决定进入 MT 或 MR 模式。如果发送 SLA+W 进入 MT 模式；如果发送 SLA+R 则进入 MR 模式。本节所提到的状态字均假设其预分频位为 "0”。

Figure 89. 主机发送模式下的数据传输



通过在 TWCR 寄存器中写入下列数值发出 START 信号：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

TWEN必须置位以使能2线串行接口，TWSTA必须置"1”来发出START 信号且 TWINT必须置 "1” 来对 TWINT 标志清零。TWI 逻辑开始检测串行总线，一旦总线空闲就发送 START。接着中断标志TWINT置位，TWSR的状态码为0x08 (见 Table91)。为进入MT模式 ，必须发送 SLA+W。这可通过对 TWDR 写入 SLA+W 来实现。完成此操作后软件清零 TWINT 标志 ， TWI 传输继续进行。这通过在 TWCR 寄存器中写入下述值完成：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

当SLA+W 发送完毕并接收到确认信号，主机的TWINT标志再次置位。此时主机的TWSR状态码可能是 0x18、 0x20 或 0x38。对各状态码的正确响应列于 Table91。

SLA+W 发送成功后可以开始发送数据包。这通过对 TWDR 写入数据实现。TWDR 只有在 TWINT 为高时方可写入。否则，访问被忽略，寄存器 TWCR 的写碰撞位 TWWC 置位。TWDR更新后，TWINT位应清零来继续传送。这通过在 TWCR寄存器中写入下述值完成:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

这过程会一直重复下去，直到最后的字节发送完且发送器产生 STOP 或 REPEATED START 信号 。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	1	X	1	0	X

REPEATED START 信号通过在 TWCR 中写入下述值实现：

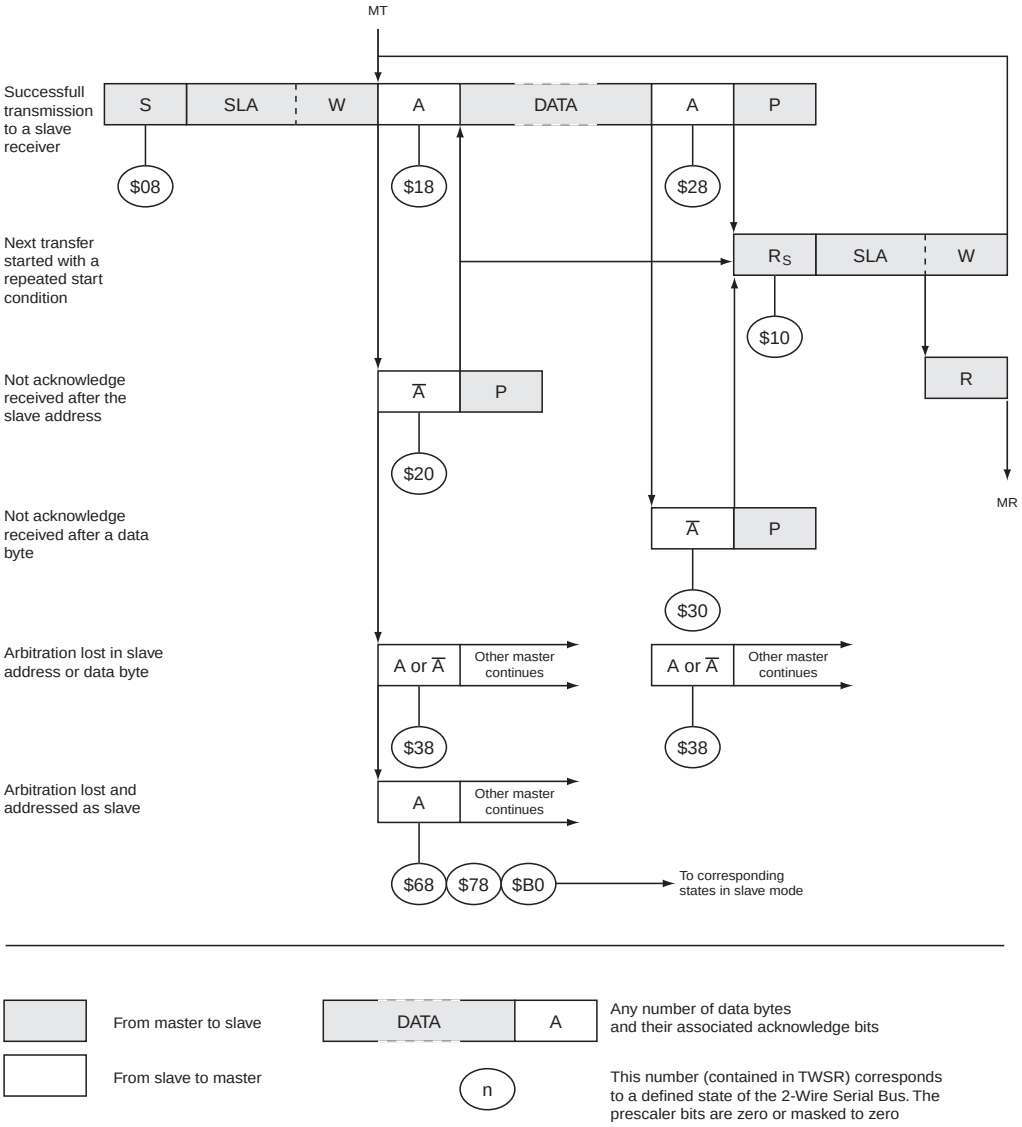
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

在 REPEATED START (状态 0x10) 后，2 线串行接口可以再次访问相同的从机，或不发送 STOP 信号来访问新的从机。 REPEATED START 使得主机可以在不丢失总线控制的条件下在从机、主机发送器及主机接收器模式间进行切换。

Table 91. 主机发送模式的状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
0x08	START 已发送	加载 SLA+W	0	0	1	X	将发送 SLA+W 将接收到 ACK 或 NOT ACK
0x10	重复 START 已发送	加载 SLA+W 或 加载 SLA+R	0 0	0 0	1 1	X X	将发送 SLA+W 将接收到 ACK 或 NOT ACK 将发送 SLA+R 切换到主机接收模式
0x18	SLA+W 已发送； 接收到 ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
0x20	SLA+W 已发送 接收到 NOT ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
0x28	数据已发送 接收到 ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
0x30	数据已发送 接收到 NOT ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
0x38	SLA+W 或数据的仲裁失败	不操作 TWDR 或 不操作 TWDR	0 1	0 0	1 1	X X	2 线串行总线将被释放，并进入未寻址从机模式 总线空闲后将发送 START

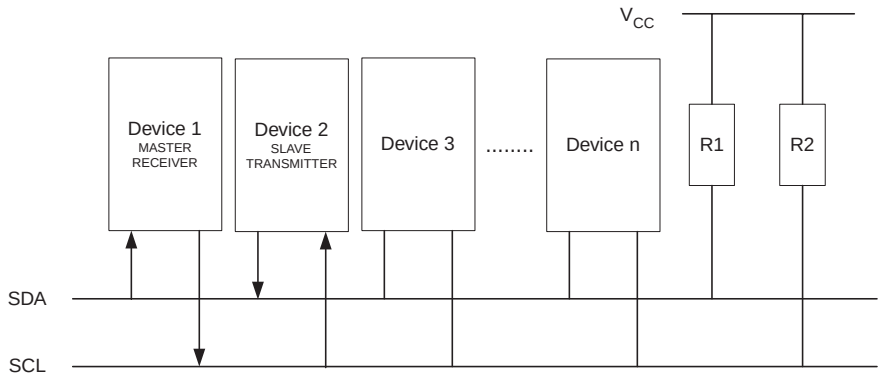
Figure 90. 主机发送模式的格式和状态



主机接收模式

在主机接收模式，主机可以从从机接收数据，如 Figure91 所示。为进入主机模式，必须发送 START 信号。紧接着的地址包格式决定进入 MT 或 MR 模式。如果发送 SLA+W 进入 MT 模式；如果发送 SLA+R 则进入 MR 模式。本节所提到的状态字均假设其预分频位为 "0”。

Figure 91. 主机接收模式下的数据传输



通过在 TWCR 寄存器中写入下列数值发出 START 信号：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

TWEN必须置位以使能2线串行接口，TWSTA必须置"1”来发出START 信号且 TWINT必须置 "1” 来对 TWINT 标志清零。TWI 逻辑开始检测串行总线，一旦总线空闲就发送 START。接着中断标志 TWINT 置位，TWSR 的状态码为 0x08 (见 Table91)。为进入 MR 模式 ，必须发送 SLA+R。这可通过对 TWDR 写入 SLA+R 来实现。完成此操作后软件清零 TWINT 标志 ， TWI 传输继续进行。这通过在 TWCR 寄存器中写入下述值完成：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

当 SLA+R 发送完毕并接收到确认信号，主机的 TWINT 标志再次置位。此时主机的 TWSR 状态码可能是 0x38、 0x40 或 0x48。对各状态码的正确响应列于 Table92。TWDR 只有在 TWINT 为高时才能读收到的数据。这过程会一直重复下去，直到最后的字节接收结束。接收完成后， MR 应通过在接收到最后的字节后发送 NACK 信号。发送器产生 STOP 或 REPEATED START 信号结束传送。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	1	X	1	0	X

REPEATED START 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

在 REPEATED START (状态 0x10) 后，2 线串行接口可以再次访问相同的从机，或不发送 STOP 信号来访问新的从机。 REPEATED START 使得主机可以在不丢失总线控制的条件下在从机、主机发送器及主机接收器模式间进行切换。

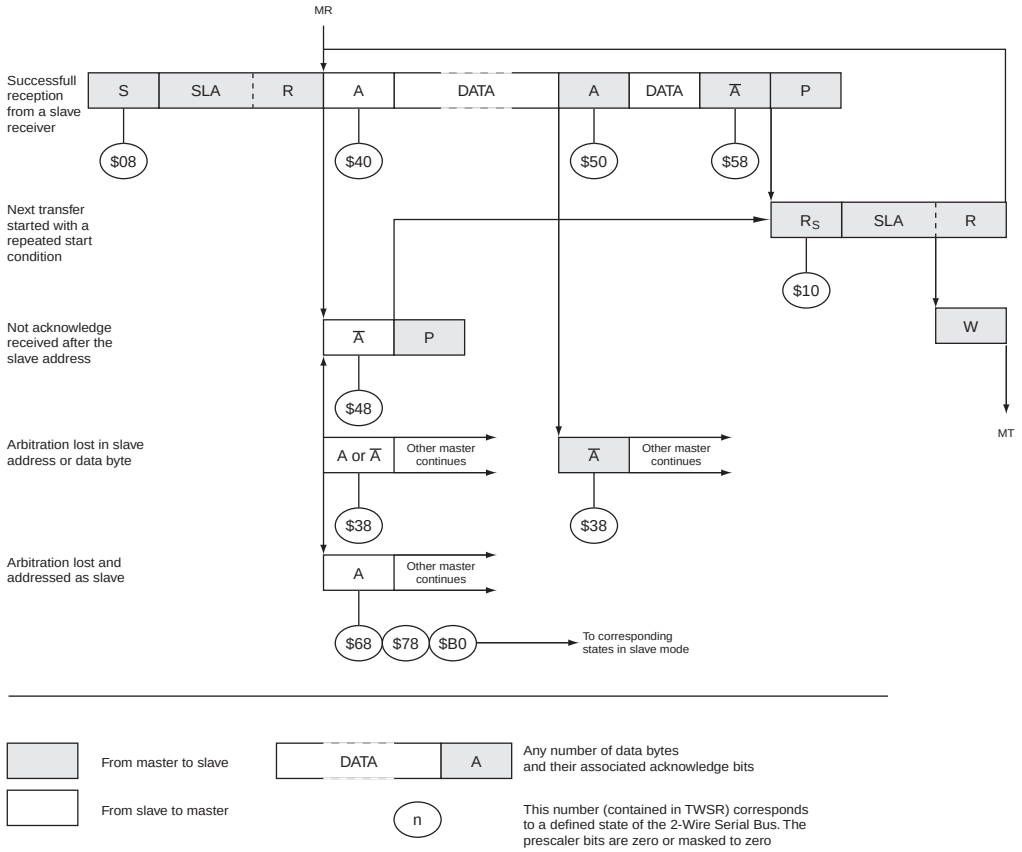
Table 92. 主机接收模式的状态码

状态码 (TWSR) 预分频位为 “0”	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	

Table 92. 主机接收模式的状态码

0x08	START 已发送	加载 SLA+R	0	0	1	X	将发送 SLA+R 将接收到 ACK 或 NOT ACK
0x10	重复 START 已发送	加载 SLA+R 或 加载 SLA+W	0 0	0 0	1 1	X X	将发送 SLA+R 将接收到 ACK 或 NOT ACK 将发送 SLA+W 逻辑切换到主机发送模式
0x38	SLA+R 或 NOT ACK 的仲裁失败	不操作 TWDR 或 不操作 TWDR	0 1	0 0	1 1	X X	2 线串行总线将被释放，并进入未寻址从机模式 总线空闲后将发送 START
0x40	SLA+R 已发送 接收到 ACK	不操作 TWDR 或 不操作 TWDR	0 0	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
0x48	SLA+R 已发送 接收到 NOT ACK	不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	1 0 1	0 1 1	1 1 1	X X X	将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
0x50	接收到数据 ACK 已返回	读数据或 读数据或	0 0	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
0x58	接收到数据 NOT ACK 已返回	读数据或 读数据或 读数据	1 0 1	0 1 1	1 1 1	X X X	将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位

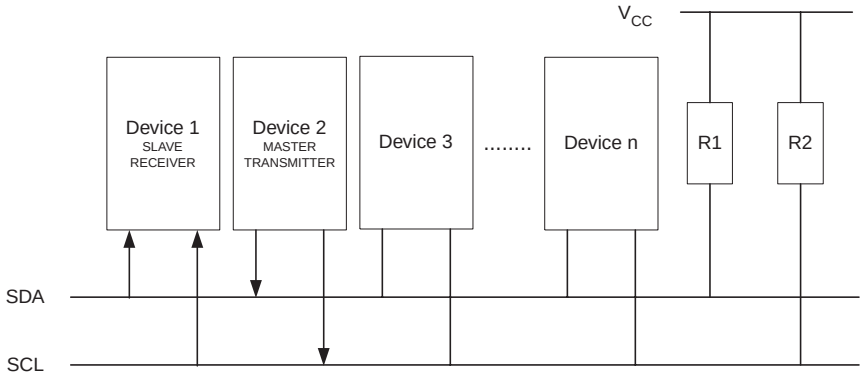
Figure 92. 主机接收模式的格式和状态



从机接收模式

在从机接收模式，从机自主机接收数据，如 Figure93 所示。本节所提到的状态字均假设其预分频位为 "0”。

Figure 93. 从机接收模式下的数据传输



为启动从机接收模式， TWAR 与 TWCR 设置如下：

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
值	器件本身从机地址							

前 7 位是主机寻址时从机响应的 TWI 接口地址。若 LSB 置位 , 则 TWI 接口响应广播地址 0x00。否则忽略广播地址。

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	0	1	0	0	0	1	0	X

TWEN必须置位以使能TWI接口。TWEA也要置位以使主机寻址到自己(从机地址或广播)时返回确认信息 ACK。 TWSTA 和 TWSTO 必须清零。

初始化 TWAR 和 TWCR 之后， TWI 接口即开始等待，直到自己的从机地址（或广播地址，如果 TWAR 的 TWGCE 置位的话) 出现在主机寻址地址当中，并且数据方向位为 0 (写)。然后 TWINT 标志置位， TWSR 则包含了相应的状态码。对各状态码的正确响应列于 Table93。当 TWI 接口处于主机模式 (状态 0x68 或 0x78) 并发生仲裁失败时 CPU 将进入从机接收模式。

如果在传输过程中 TWEA 复位， TWI 接口在接收到下一个字节后将向 SDA 返回“ 无应答” 。 TWEA 复位时 TWI 接口不再响应自己的从机地址，但是会继续监视总线。一旦TWEA置位就可以恢复地址识别和响应。也就是说，可以利用TWEA暂时将TWI接口从总线中隔离出来。

在除空闲模式外的其它休眠模式时， TWI 接口的时钟被关闭。若使能了从机接收模式，接口将利用总线时钟继续响应广播地址 / 从机地址。地址匹配将唤醒 CPU。在唤醒期间，TWI接口将保持SCL为低电平，直至TWCINT标志清零。当AVR时钟恢复正常运行后TWI可以接收更多的数据。显然如果 AVR 设置为长启动时间，时钟线 SCL 可能会长时间保持低，阻塞其它数据的传送。

当 MCU 从这些休眠模式唤醒时，和正常工作模式不同的是，数据寄存器 TWDR 的数据并不反映总线上出现的最后一个字节。

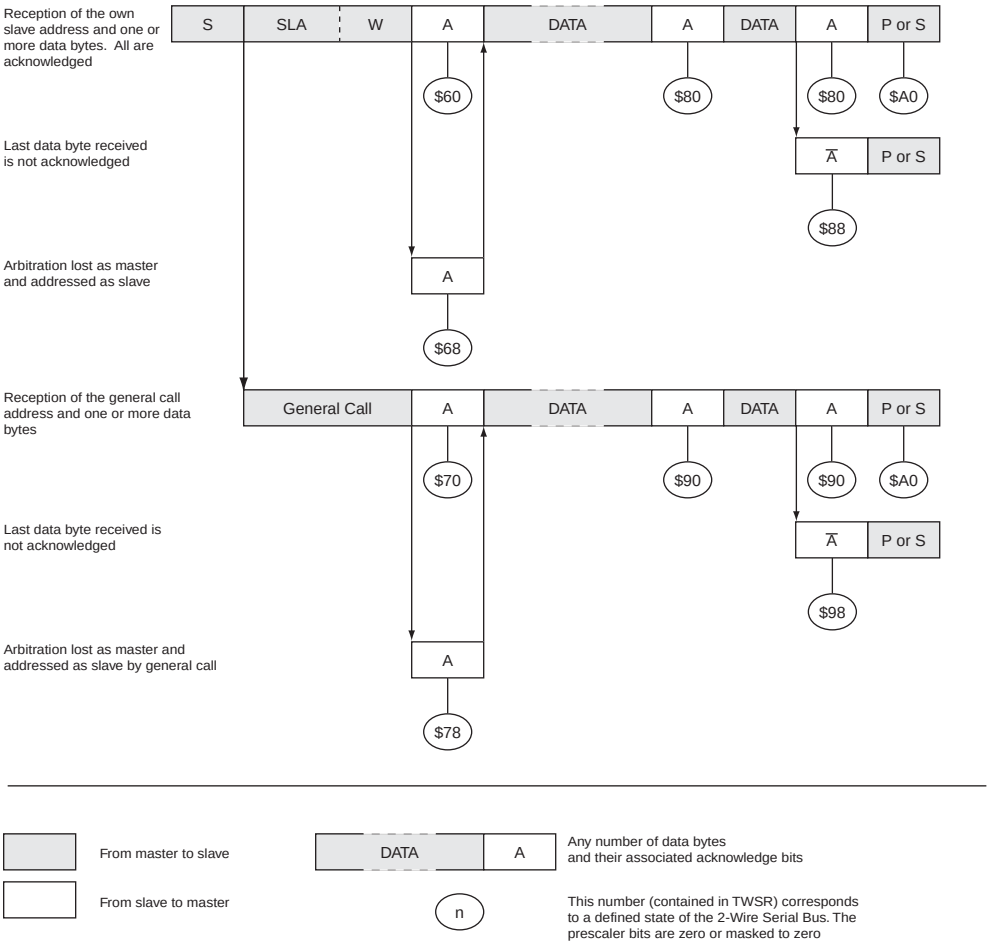
Table 93. 从机接收模式的状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
			对 TWCR 的操作				
		读 / 写 TWDR	STA	STO	TWINT	TWEA	
0x60	自己的 SLA+W 已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
0x68	SLA+R/W 作为主机的仲裁失败；自己的 SLA+W 已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
0x70	接收到广播地址 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
0x78	SLA+R/W 作为主机的仲裁失败；接收到广播地址 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
0x80	以前以自己的 SLA+W 被寻址；数据已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
0x88	以前以自己的 SLA+W 被寻址；数据已经被接收 NOT ACK 已返回	读数据或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		读数据或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别
		读数据或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		读数据或	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START

Table 93. 从机接收模式的状态码

0x90	以前以广播方式被寻址；数据已经被接收 ACK 已返回	读数据或 读数据	X X	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
0x98	以前以广播方式被寻址；数据已经被接收 NOT ACK 已返回	读数据或 读数据或 r 读数据或 读数据	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START
0xA0	在以从机工作时接收到 STOP 或重复 START	读数据或 读数据或 r 读数据或 读数据	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START

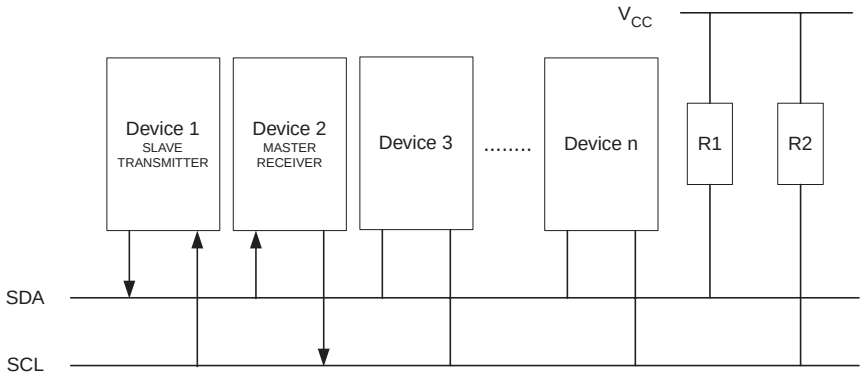
Figure 94. 从机接收模式的格式和状态



从机发送模式

在从机发送模式，从机可以向主机发送数据，如 Figure95 所示。本节所提到的状态字均假设其预分频位为 "0”。

Figure 95. 从机发送模式下的数据传输



为启动从机发送模式， TWAR 与 TWCR 设置如下：

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
值	器件本身从机地址							

前 7 位是主机寻址时从机响应的 TWI 接口地址。若 LSB 置位，则 TWI 接口响应广播地址 0x00。否则忽略广播地址。

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	0	1	0	0	0	1	0	X

TWEN必须置位以使能TWI接口。TWEA也要置位以便主机寻址到自己(从机地址或广播)时返回确认信息 ACK。 TWSTA 和 TWSTO 必须清零。

初始化 TWAR 和 TWCR 之后， TWI 接口即开始等待，直到自己的从机地址（或广播地址，如果 TWAR 的 TWGCE 置位的话）出现在主机寻址地址当中，并且数据方向位为 "1”（读）。然后 TWI 中断标志置位，TWSR 则包含了相应的状态码。对各状态码的正确响应列于 Table94。当 TWI 接口处于主机模式（状态 0xB0) 并发生仲裁失败时 CPU 将进入从机发送模式。

如果在传输过程中 TWEA 复位，TWI 接口发送完数据之后进入状态 0xC0 或 0xC8。接口也切换到未寻址从机模式，忽略任何后续总线传输。从而主机接收到的数据全为 “1”。如果主机需要附加数据位（通过发送 ACK)，即使从机已经传送结束，也进入状态 0xC8。

TWEA复位时TWI接口不再响应自己的从机地址，但是会继续监视总线。一旦TWEA置位就可以恢复地址识别和响应。也就是说，可以利用 TWEA 暂时将 TWI 接口从总线中隔离出来。

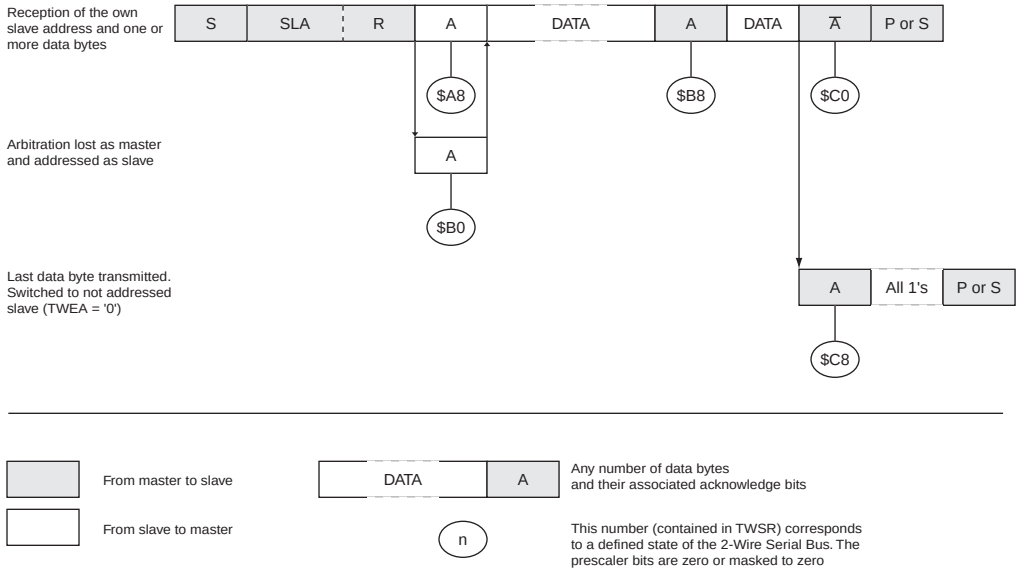
在除空闲模式外的其它休眠模式时， TWI 接口的时钟被关闭。若使能了从机接收模式，接口将利用总线时钟继续响应广播地址 / 从机地址。地址匹配将唤醒 CPU。在唤醒期间，TWI接口将保持SCL为低电平，直至TWCINT标志清零。当AVR时钟恢复正常运行后可以发送更多的数据。显然如果 AVR 设置为长启动时间，时钟线 SCL 可能会长时间保持低，阻塞其它数据的传送。

当 MCU 从这些休眠模式唤醒时，和正常工作模式不同的是，数据寄存器 TWDR 的数据并不反映总线上出现的最后一个字节。

Table 94. 从机发送模式的状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
			对 TWCR 的操作				
		读 / 写 TWDR	STA	STO	TWINT	TWEA	
0xA8	自己的 SLA+R 已经被接收 ACK 已返回	加载一字节的数据或	X	0	1	0	发送一字节的数据，接收 NOT ACK
		加载一字节的数据	X	0	1	1	发送数据，接收 ACK
0xB0	SLA+R/W 作为主机的仲裁失败；自己的 SLA+R 已经被接收 ACK 已返回	加载一字节的数据或	X	0	1	0	发送一字节的数据，接收 NOT ACK
		加载一字节的数据	X	0	1	1	发送数据，接收 ACK
0xB8	TWDR 里数据已经发送 接收到 ACK	加载一字节的数据或	X	0	1	0	发送一字节的数据，接收 NOT ACK
		加载一字节的数据	X	0	1	1	发送数据，接收 ACK
0xC0	TWDR 里数据已经发送 接收到 NOT ACK	不操作 TWDR 或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		不操作 TWDR 或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 GC = “1”，GCA 也可以识别
		不操作 TWDR 或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		不操作 TWDR	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 GC = “1”，GCA 也可以识别；总线空闲时发送 START
0xC8	TWDR 的一字节数据已经发送 (TWAE = “0”); 接收到 ACK	不操作 TWDR 或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		不操作 TWDR 或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 GC = “1”，GCA 也可以识别
		不操作 TWDR 或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		不操作 TWDR	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 GC = “1”，GCA 也可以识别；总线空闲时发送 START

Figure 96. 从机发送模式的格式和状态



其它状态

有两个状态码没有相应的 TWI 状态定义，见 Table95。

状态 0xF8 表明当前没有相关信息，因为中断标志 TWINT 为 "0”。这种状态可能发生在自己的 TWI 接口没有参与串行传输的时候。

状态 0x00 表示在串行传输过程中发生了总线错误。当 START 或 STOP 出现在错误的位置时总线错误就会发生。比如说在地址和数据、地址和ACK之间出现了START或STOP。总线错误将导致 TWINT 置位。为了从错误中恢复出来，必须置位标志 TWSTO，并通过写 "1” 以清零 TWINT。这将导致 TWI 接口进入未寻址从机模式、标志 TWSTO 被清零 (TWCR 的其他位不受影响)，以及 SDA 和 SCL 被释放，但是不会产生 STOP。

Table 95. 其它状态码

状态码 (TWSR) 预分频位为 ”0”	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
0xF8	没有相关的状态信息；TWINT = “0”	不操作 TWDR	No TWCR action				等待或进行当前传输
0x00	由于非法的 START 或 STOP 引起的总线错误	不操作 TWDR	0	1	1	X	只影响内部硬件；不会发送 STOP 到总线上。总线将释放并清零 TWSTO

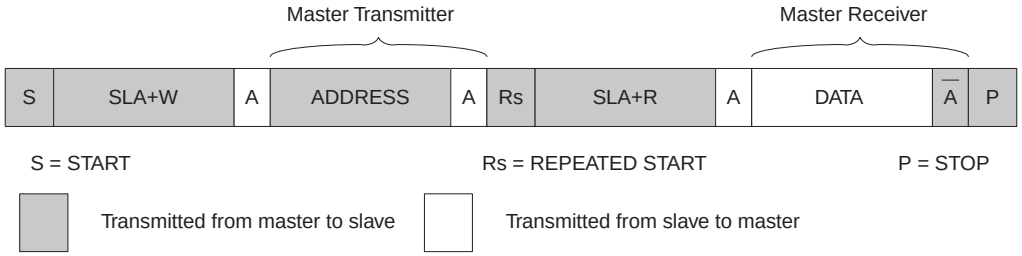
几种 TWI 模式的组合

在某些情况下，为完成期望的工作，必须将几种TWI模式组合起来。例如从串行EEPROM读取数据。典型的这种传输包括以下步骤：

1. 传输必须启动
2. 必须告诉 EEPROM 读取的位置
3. 必须完成读操作
4. 传送必须结束

注意数据可从主机传到从机，反之也可。首先主机必须告诉读从机读取实际的位置，因此需要使用 MT 模式；然后数据必须由从机读出，需要使用 MR 模式，但传送方向必须改变。在上述步骤中，主机必须保持对总线的控制，且以上各步骤应该自动进行。如果在多主机系统中违反这一规则，即在第二步与第三步之间其它主机改变 EEPROM 中的数据指针，则主机读取的数据位置是错误的。传送方向改变是通过在发送地址字节与接收数据之间发送 REPEATED START 信号来实现的。在发送 REPEATED START 信号后，主机继续保持总线的控制权。下图给出传送的流程图。

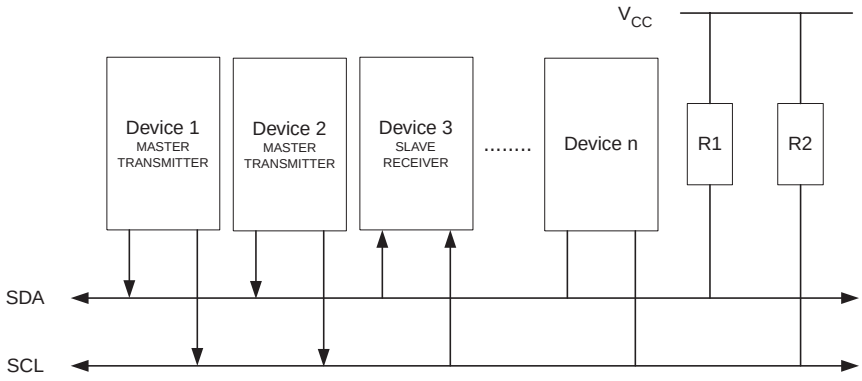
Figure 97. 几种 TWI 模式联合访问串行 EEPROM



多主机系统与仲裁

如果有多个主机连接在同一总线上，它们中的一个或多个也许会同时开始一个数据传送。TWI 协议确保在这种情况下，通过一个仲裁过程，允许其中的一个主机进行传送而不会丢失数据。总线仲裁的例子如下所述，该例中有两个主机试图向从接收器发送数据。

Figure 98. 仲裁示例



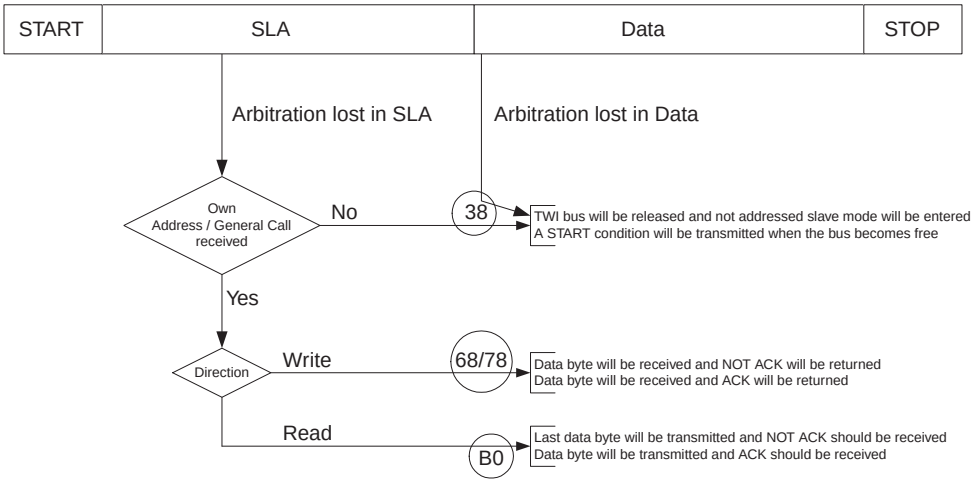
有几种不同的情况会产生总线仲裁过程：

- 两个或更多的主机同时与一个从机进行通信。在这种情况下，无论主机或从机都不知道有总线的竞争。
- 两个或更多的主机同时对同一个从机进行不同的数据或方向的访问。在这种情况下，会在 READ/WRITE 位或数据间发生仲裁。主机试图在 SDA 线上输出一个高电平时，如果其他主机已经输出 "0”，则该主机在总线仲裁中失败。失败的主机将转换成未被寻址的从机模式，或等待总线空闲后发送一个新的 START 信号，这由应用程序决定。

- 两个或更多的主机访问不同的从机。在这种情况下，总线仲裁在 SLA 发生。主机试图在 SDA 线上输出一个高电平时，如有其它主机已经输出 "0”，则该主机将在总线仲裁中失败。在 SLA 总线仲裁失败的主机将切换到从机模式，并检查自己是否被获得总线控制权的主机寻址。如果被寻址，它将进入 SR 或 ST 模式，这取决于 SLA 的 READ/WRITE 位的值。如果它未被寻址，将转换到未被寻址的从机模式或等待总线空闲，发送一个新的 START 信号，这由应用程序决定。

Figure99 描述了总线仲裁的过程，图中的数字为 TWI 的状态值。

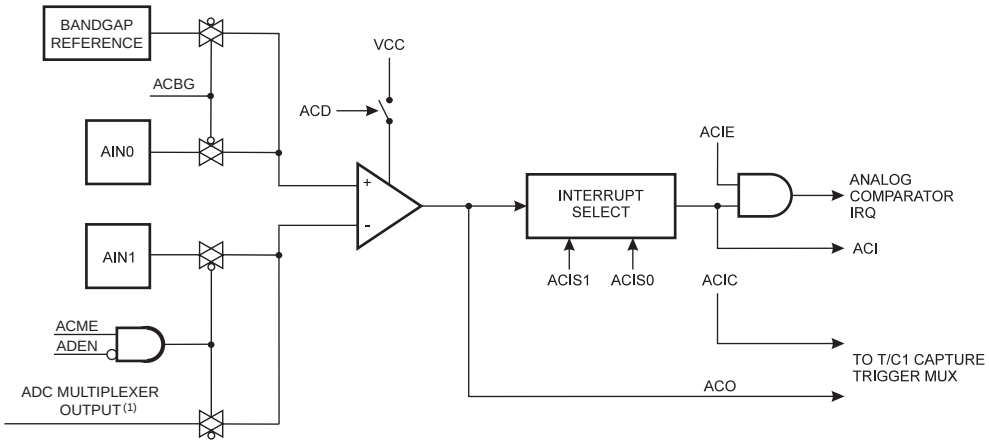
Figure 99. 总线仲裁过程



模拟比较器

模拟比较器对正极 AIN0 的值与负极 AIN1 的值进行比较。当 AIN0 上的电压比负极 AIN1 上的电压要高时，模拟比较器的输出 ACO 即置位。比较器的输出可用来触发定时器 / 计数器 1 的输入捕捉功能。此外，比较器还可触发自己专有的、独立的中断。用户可以选择比较器是以上升沿、下降沿还是交替变化的边沿来触发中断。Figure100 为比较器及其外围逻辑电路的框图。

Figure 100. 模拟比较器框图 (2)



Notes: 1. 见 P215Table97 。
2. 模拟比较器的管脚分布见 P2Figure 1 及 P71Table40

ADC 控制及状态寄存器 B—ADCSRB

位	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
读 / 写	R	R/W	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 6 – ACME: 模拟比较器多路复用器使能**

当此位为逻辑 "1”，且 ADC 处于关闭状态 (ADCSRA 寄存器的 ADEN 为 "0”) 时，ADC 多路复用器为模拟比较器选择负极输入。当此位为 "0” 时， AIN1 连接到比较器的负极输入端。更详细描述的请参见 P216” 模拟比较器多路输入 ”。

模拟比较器控制及状态寄存器—ACSR

位	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	N/A	0	0	0	0	0	

- **位 7 – ACD: 模拟比较器禁用**

ACD 置位时，模拟比较器的电源被切断。可以在任何时候设置此位来关掉模拟比较器。这可以减少器件工作模式及空闲模式下的功耗。改变 ACD 位时，必须清零 ACSR 寄存器的 ACIE 位来禁止模拟比较器中断。否则 ACD 改变时可能会产生中断。

- **位 6 – ACBG: 选择模拟比较器的能隙基准源**

ACBG 置位后，模拟比较器的正极输入由能隙基准源所取代。否则， AIN0 连接到模拟比较器的正极输入。见 P43“ 片内基准电压 ”。

- **位 5 – ACO: 模拟比较器输出**

模拟比较器的输出经过同步后直接连到 ACO。同步机制引入了 1-2 个时钟周期的延时。

- **位 4 – ACI: 模拟比较器中断标志**

当比较器的输出事件触发了由 ACIS1 及 ACIS0 定义的中断模式时 ,ACI 置位。如果 ACIE 和 SREG 寄存器的全局中断标志 I 也置位 ,那么模拟比较器中断服务程序即得以执行 ,同时 ACI 被硬件清零。 ACI 也可以通过写 "1” 来清零。

- **位 3 – ACIE: 模拟比较器中断使能**

当 ACIE 位被置 "1” 且状态寄存器中的全局中断标志 I 也被置位时，模拟比较器中断被激活。否则中断被禁止。

- **位 2 – ACIC: 模拟比较器输入捕捉使能**

ACIC 置位后允许通过模拟比较器来触发 T/C1 的输入捕捉功能。此时比较器的输出被直接连接到输入捕捉的前端逻辑，从而使得比较器可以利用 T/C1 输入捕捉中断逻辑的噪声抑制器及触发沿选择功能。 ACIC 为 "0” 时模拟比较器及输入捕捉功能之间没有任何联系。为了使比较器可以触发 T/C1 的输入捕捉中断，定时器中断屏蔽寄存器 TIMSK1 的 ICIE1 必须置位。

- **位 1, 0 – ACIS1, ACIS0: 模拟比较器中断模式选择**

这两位确定触发模拟比较器中断的事件。 Table96 给出了不同的设置。

Table 96. ACIS1/ACIS0 设置

ACIS1	ACIS0	中断模式
0	0	比较器输出变化即可触发中断
0	1	保留
1	0	比较器输出的下降沿产生中断
1	1	比较器输出的上升沿产生中断

需要改变 ACIS1/ACIS0 时，必须清零 ACSR 寄存器的中断使能位来禁止模拟比较器中断。否则有可能在改变这两位时产生中断。

模拟比较器多路输入

可以选择 ADC7..0 之中的任意一个来代替模拟比较器的负极输入端。ADC 复用器可用来完成这个功能。当然，为了使用这个功能首先必须关掉 ADC。如果模拟比较器复用器使能位 (ADCSRB 中的 ACME) 被置位，且 ADC 也已经关掉 (ADCSRA 寄存器的 ADEN 为 0) ,则可以通过 ADMUX 寄存器的 MUX2..0 来选择替代模拟比较器负极输入的管脚 ,详见 Table97。如果 ACME 清零或 ADEN 置位，则模拟比较器的负极输入为 AIN1。

Table 97. 模拟比较器复用输入

ACME	ADEN	MUX2..0	模拟比较器负极输入
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

数字输入禁止寄存器 1—DIDR1

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	AIN1D	AIN0D	DIDR1
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7..2 – Res: 保留位**

ATmega48/88/168 中的这几位未使用，读返回值为 "0”。

- **位 1, 0 – AIN1D, AIN0D: AIN1, AIN0 数字输入禁止**

AIN1D和AIN0D置"1”后 ,AIN1/0引脚的数字输入缓冲器被禁止 ,相应的PIN寄存器的读返回值为 "0”。当 AIN1/0 引脚加载了模拟信号，且当前应用不需要 AIN1/0 引脚的数字输入缓冲器时， AIN1D 和 AIN0D 应该置位以降低数字输入缓冲的功耗。

模数转换器

特点

- **10 位 精度**
- **0.5 LSB 的非线性度**
- **± 2 LSB 的绝对精度**
- **65 - 260 µs 的转换时间**
- **最高分辨率时采样率高达 15 kSPS**
- **6 路复用的单端输入通道**
- **2 路附加的复用单端输入通道 (TQFP 与 MLF 封装)**
- **可选的向左调整 ADC 读数**
- **0 - V_{CC} 的 ADC 输入电压范围**
- **可选的 1.1V ADC 参考电压**
- **连续转换或单次转换模式**
- **ADC 转换结束中断**
- **基于睡眠模式的噪声抑制器**

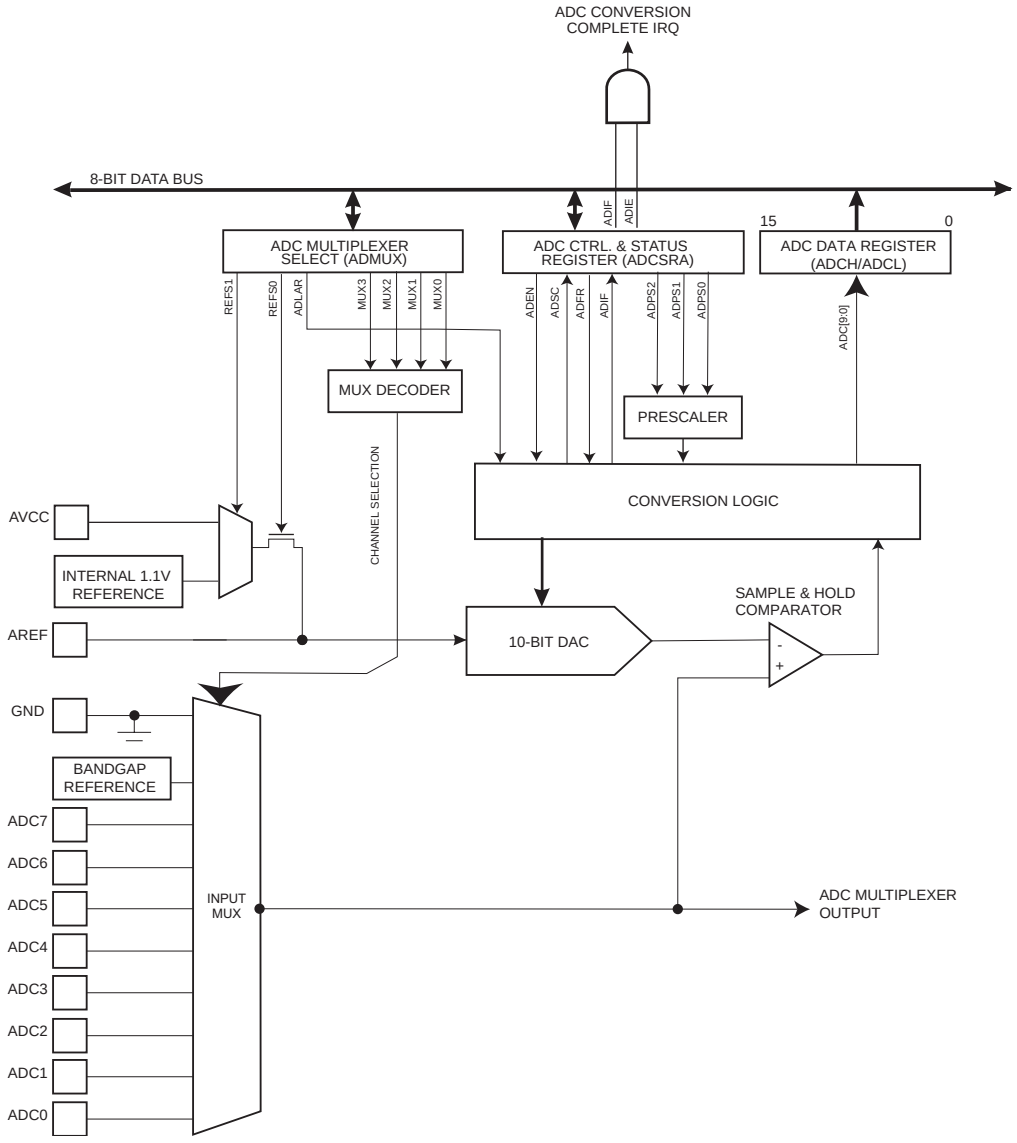
ATmega48/88/168 有一个10位的逐次逼近型ADC。ADC与一个8通道的模拟多路复用器连接，能对来自端口A的8路单端输入电压进行采样。单端电压输入以0V (GND)为基准。

ADC 包括一个采样保持电路，以确保在转换过程中输入到 ADC 的电压保持恒定。ADC 的框图如 Figure101 所示。

ADC 由 AVCC 引脚单独提供电源。 AVCC 与 V_{CC} 之间的偏差不能超过 ± 0.3V。请参考 P223”ADC 噪声抑制器 ” 来了解如何连接这个引脚。

标称值为 1.1V 的基准电压，以及 AVCC，都位于器件之内。基准电压可以通过在 AREF 引脚上加一个电容进行解耦，以更好地抑制噪声。

Figure 101. 模数转换器方框图



ADC 通过逐次逼近的方法将输入的模拟电压转换成一个 10 位的数字量。最小值代表 GND，最大值代表 AREF 引脚上的电压再减去 1 LSB。通过写 ADMUX 寄存器的 REFSn 位可以把 AVCC 或内部 1.1V 的参考电压连接到 AREF 引脚。在 AREF 上外加电容可以对片内参考电压进行解耦以提高噪声抑制性能。

模拟输入通道可以通过写 ADMUX 寄存器的 MUX 位来选择。任何 ADC 输入引脚，像 GND 及固定能隙参考电压，都可以作为 ADC 的单端输入。通过设置 ADCSRA 寄存器的 ADEN 即可启动 ADC。只有当 ADEN 置位时参考电压及输入通道选择才生效。 ADEN 清零时 ADC 并不耗电，因此建议在进入节能睡眠模式之前关闭 ADC。

ADC 转换结果为 10 位，存放于 ADC 数据寄存器 ADCH 及 ADCL 中。默认情况下转换结果为右对齐，但可通过设置 ADMUX 寄存器的 ADLAR 变为左对齐。

如果要求转换结果左对齐，且最高只需 8 位的转换精度，那么只要读取 ADCH 就足够了。否则要先读 ADCL，再读 ADCH，以保证数据寄存器中的内容是同一次转换的结果。一旦读出 ADCL， ADC 对数据寄存器的寻址就被阻止了。也就是说，读取 ADCL 之后，即使

在读 ADCH 之前又有一次 ADC 转换结束，数据寄存器的数据也不会更新，从而保证了转换结果不丢失。 ADCH 被读出后， ADC 即可再次访问 ADCH 及 ADCL 寄存器。

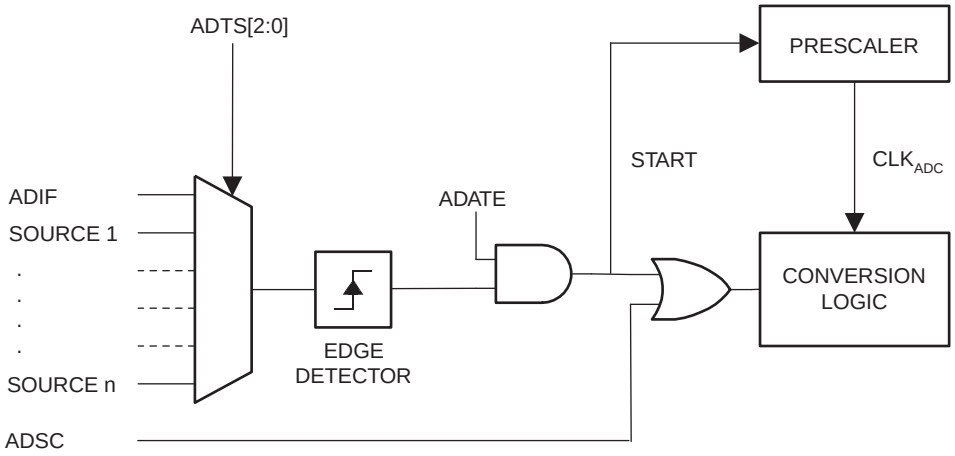
ADC转换结束可以触发中断。即使由于转换发生在读取ADCH与ADCL之间而造成ADC无法访问数据寄存器，并因此丢失了转换数据，中断仍将触发。

启动一次转换

向 ADC 启动转换位 ADSC 位写 "1" 可以启动单次转换。在转换过程中此位保持为高，直到转换结束，然后被硬件清零。如果在转换过程中选择了另一个通道，那么 ADC 会在改变通道前完成这一次转换。

ADC转换有不同的触发源。设置ADCSRA寄存器的ADC自动触发允许位ADATE可以使能自动触发。设置 ADCSRB 寄存器的 ADC 触发选择位 ADTS 可以选择触发源 (见触发源列表中对 ADTS 的描述)。当所选的触发信号产生上跳沿时， ADC 预分频器复位并开始转换。这提供了一个在固定时间间隔下启动转换的方法。转换结束后即使触发信号仍然存在，也不会启动一次新的转换。如果在转换过程中触发信号中又产生了一个上跳沿，这个上跳沿将被忽略。即使特定的中断被禁止或全局中断使能位为 0，中断标志仍将置位。这样可以在不产生中断的情况下触发一次转换。但是为了在下次中断事件发生时触发新的转换，必须将中断标志清零。

Figure 102. ADC 自动触发逻辑

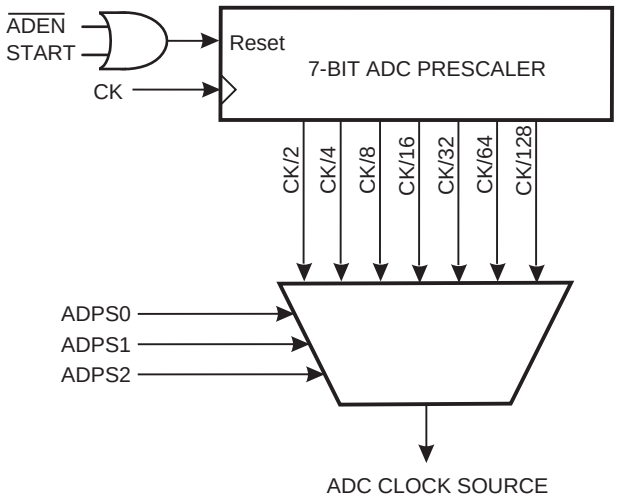


使用 ADC 中断标志作为触发源，可以在正在进行的转换结束后即开始下一次 ADC 转换。之后 ADC 便工作在连续转换模式，持续地进行采样并对 ADC 数据寄存器进行更新。第一次转换通过向 ADCSRA 寄存器的 ADSC 写 1 来启动。在此模式下，后续的 ADC 转换不依赖于 ADC 中断标志 ADIF 是否置位。

如果使能了自动触发，置位 ADCSRA 寄存器的 ADSC 将启动单次转换。 ADSC 标志还可用来检测转换是否在进行之中。不论转换是如何启动的，在转换进行过程中 ADSC 一直为 "1"。

预分频及 ADC 转换时序

Figure 103. ADC 预分频器



在默认条件下，逐次逼近电路需要一个从50 kHz到200 kHz的输入时钟以获得最大精度。如果所需的转换精度低于 10 比特，那么输入时钟频率可以高于 200 kHz，以达到更高的采样率。

ADC 模块包括一个预分频器，它可以由任何超过 100 kHz 的 CPU 时钟来产生可接受的 ADC 时钟。预分频器通过 ADCSRA 寄存器的 ADPS 进行设置。置位 ADCSRA 寄存器的 ADEN 将使能 ADC，预分频器开始计数。只要 ADEN 为 1，预分频器就持续计数，直到 ADEN 清零。

ADCSRA 寄存器的 ADSC 置位后，单端转换在下一个 ADC 时钟周期的上升沿开始启动。

正常转换需要 13 个 ADC 时钟周期。为了初始化模拟电路， ADC 使能 (ADCSRA 寄存器的 ADEN 置位) 后的第一次转换需要 25 个 ADC 时钟周期。

在普通的 ADC 转换过程中，采样保持在转换启动之后的 1.5 个 ADC 时钟开始；而第一次 ADC 转换的采样保持则发生在转换启动之后的 13.5 个 ADC 时钟。转换结束后，ADC 结果被送入 ADC 数据寄存器，且 ADIF 标志置位。ADSC 同时清零 (单次转换模式)。之后软件可以再次置位 ADSC 标志，从而在 ADC 的第一个上升沿启动一次新的转换。

使用自动触发时，触发事件发生将复位预分频器。这保证了触发事件和转换启动之间的延时是固定的。在此模式下，采样保持在触发信号上升沿之后的 2 个 ADC 时钟发生。为了实现同步逻辑需要额外的 3 个 CPU 时钟周期。

在连续转换模式下，当 ADSC 为 1 时，只要转换一结束，下一次转换马上开始。转换时间请见 Table98。

Figure 104. ADC 时序图，第一次转换 (单次转换模式)

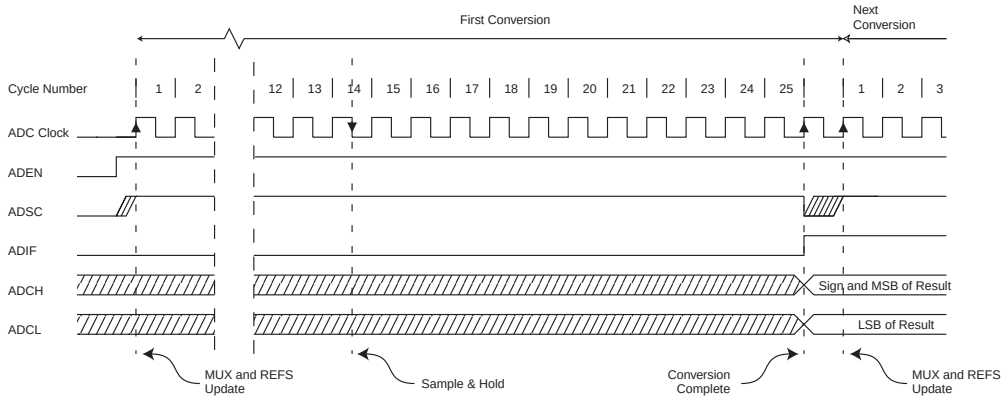


Figure 105. ADC 时序图，单次转换

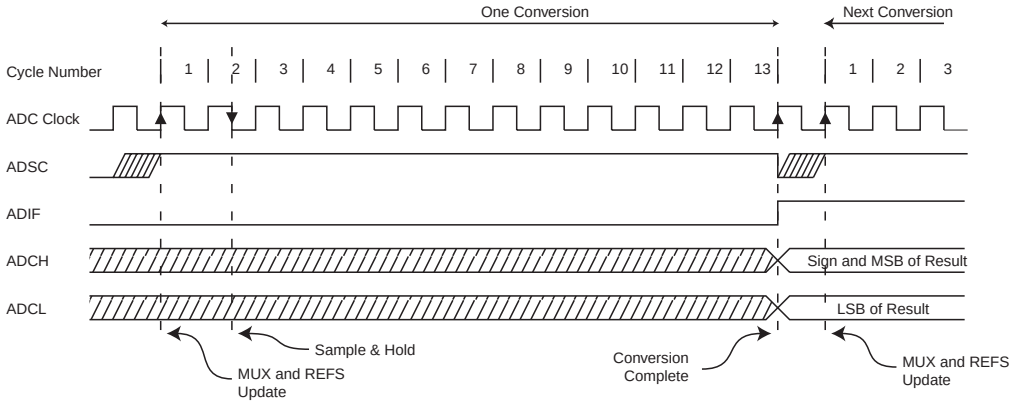


Figure 106. ADC 时序图，自动触发的转换

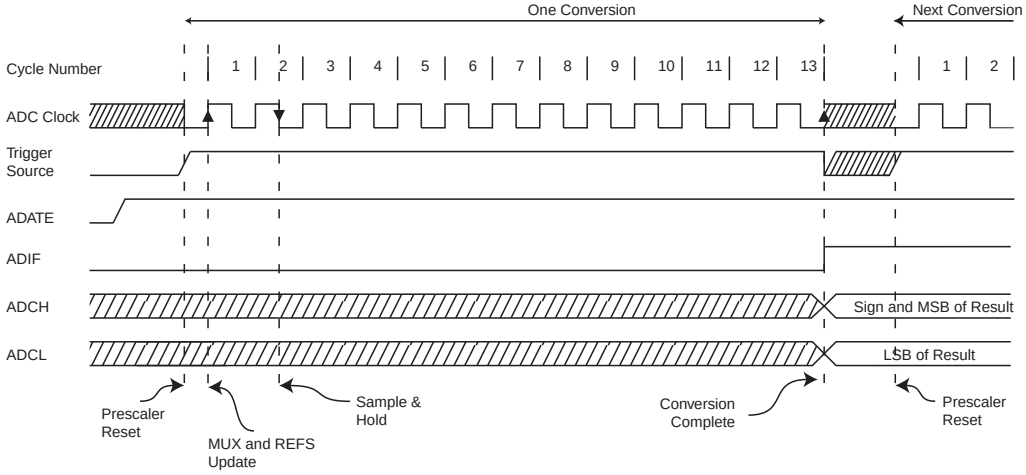


Figure 107. ADC 时序图，连续转换

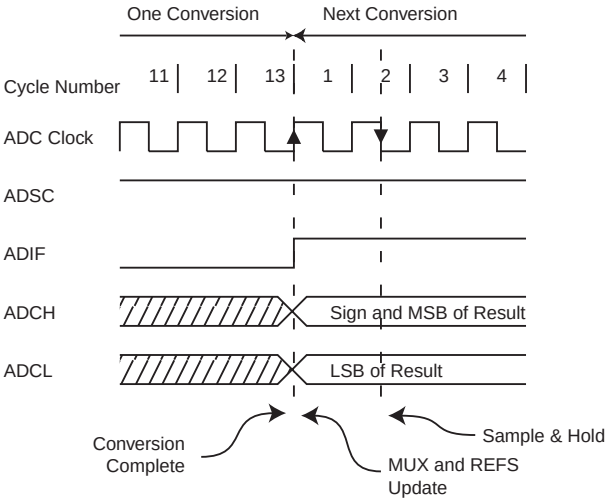


Table 98. ADC 转换时间

条件	采样 & 保持 (启动转换后的时钟周期数)	转换时间 (周期)
第一次转换	14.5	25
正常转换，单端	1.5	13
自动触发的转换	2	13.5

改变通道及基准源

ADMUX寄存器中的MUXn及REFS1:0通过临时寄存器实现了单缓冲。CPU可对此临时寄存器进行随机访问。这保证了在转换过程中通道和基准源的切换发生于安全的时刻。在转换启动之前通道及基准源的选择可随时进行。一旦转换开始就不允许再选择通道和基准源了，从而保证 ADC 有充足的采样时间。在转换完成 (ADCSRA 寄存器的 ADIF 置位) 之前的最后一个时钟周期，通道和基准源的选择又可以重新开始。转换的开始时刻为 ADSC 置位后的下一个时钟的上升沿。因此，建议用户在置位 ADSC 之后的一个 ADC 时钟周期里，不要操作 ADMUX 以选择新的通道及基准源。

使用自动触发时，触发事件发生的时间是不确定的。为了控制新设置对转换的影响，在更新 ADMUX 寄存器时一定要特别小心。

若ADATE及ADEN都置位，则中断事件可以在任意时刻发生。如果在此期间改变ADMUX寄存器的内容，那么用户就无法判别下一次转换是基于旧的设置还是最新的设置。在以下时刻可以安全地对 ADMUX 进行更新：

1. ADATE 或 ADEN 为 "0”
2. 在转换过程中，但是在触发事件发生后至少一个 ADC 时钟周期
3. 转换结束之后，但是在作为触发源的中断标志清零之前

如果在上面提到的任一种情况下更新 ADMUX，那么新设置将在下一次 ADC 时生效。

ADC 输入通道

选择模拟通道时请注意以下事项：

工作于单次转换模式时，总是在启动转换之前选定通道。在 ADSC 置位后的一个 ADC 时钟周期就可以选择新的模拟输入通道了。但是最简单的办法是等待转换结束后再改变通道。

在连续转换模式下，总是在第一次转换开始之前选定通道。在 ADSC 置位后的一个 ADC 时钟周期就可以选择新的模拟输入通道了。但是最简单的办法是等待转换结束后再改变通道。然而，此时新一次转换已经自动开始了，下一次的转换结果反映的是以前选定的模拟输入通道。以后的转换才是针对新通道的。

ADC 参考电压源

ADC的参考电压源(V_{REF})反映了ADC的转换范围。若单端通道电平超过了V_{REF}，其结果将接近 0x3FF。 V_{REF} 可以是 AVCC、内部 1.1V 基准或外接于 AREF 引脚的电压。

AVCC通过一个无源开关与ADC相连。片内的1.1V参考电压由能隙基准源(V_{BG})通过内部放大器产生。无论是哪种情况，AREF 都直接与 ADC 相连，通过在 AREF 与地之间外加电容可以提高参考电压的抗噪性。 V_{REF} 可通过高输入内阻的伏特表在 AREF 引脚测得。由于 V_{REF} 的阻抗很高，因此只能连接容性负载。

如果将一个固定电源接到 AREF 引脚，那么用户就不能选择其他的基准源了，因为这会导致片内基准源与外部参考源的短路。如果 AREF 引脚没有联接任何外部参考源，用户可以选择 AVCC 或 1.1V 作为基准源。参考源改变后的第一次 ADC 转换结果可能不准确，建议用户不要使用这一次的转换结果。

ADC 噪声抑制器

ADC的噪声抑制器使其可以在睡眠模式下进行转换，从而降低由于CPU及外围I/O设备噪声引入的影响。噪声抑制器可在 ADC 降噪模式及空闲模式下使用。为了使用这一特性，应采用如下步骤：

1. 确定 ADC 已经使能，且没有处于转换状态。工作模式应该为单次转换，并且 ADC 转换结束中断使能。
2. 进入 ADC 降噪模式 (或空闲模式)。一旦 CPU 被挂起， ADC 便开始转换。
3. 如果在ADC转换结束之前没有其他中断产生，那么ADC中断将唤醒CPU并执行 ADC 转换结束中断服务程序。如果在 ADC 转换结束之前有其他的中断源唤醒了 CPU，对应的中断服务程序得到执行。 ADC 转换结束后产生 ADC 转换结束中断请求。 CPU 将工作到新的休眠指令得到执行。

进入除空闲模式及 ADC 降噪模式之外的其他休眠模式时，ADC 不会自动关闭。在进入这些休眠模式时，建议将 ADEN 清零以降低功耗。

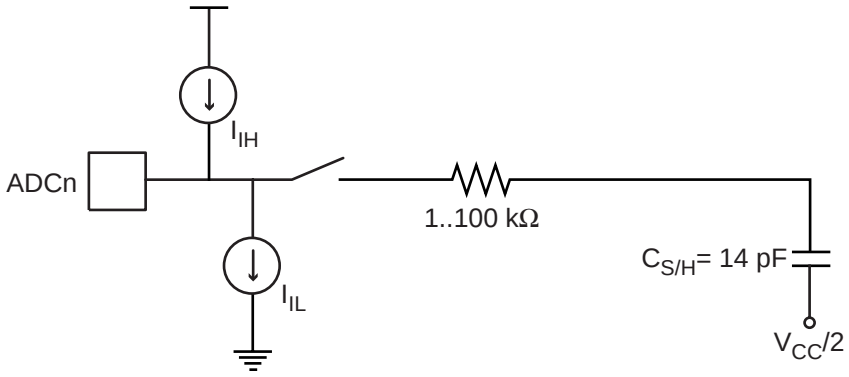
模拟输入电路

单端通道的模拟输入电路见 Figure 108.。不论是否用作 ADC 的输入通道，输入到 ADCn 的模拟信号都受到引脚电容及输入泄露的影响。用作 ADC 的输入通道时，模拟信号源必须通过一个串联电阻 (输入通道的组合电阻) 驱动采样保持 (S/H) 电容。

ADC针对那些输出阻抗接近于10 kΩ或更小的模拟信号做了优化。对于这样的信号采样时间可以忽略不计。若信号具有更高的阻抗，那么采样时间就取决于对 S/H 电容充电的时间。这个时间可能变化很大。建议用户使用输出阻抗低且变化缓慢的模拟信号，因为这可以减少对 S/H 电容的电荷传输。

频率高于奈奎斯特频率 ($f_{ADC}/2$) 的信号源不能用于任何一个通道，这样可以避免不可预知的信号卷积造成的失真。在把信号输入到 ADC 之前最好使用一个低通滤波器来滤掉高频信号。

Figure 108. 模拟输入电路

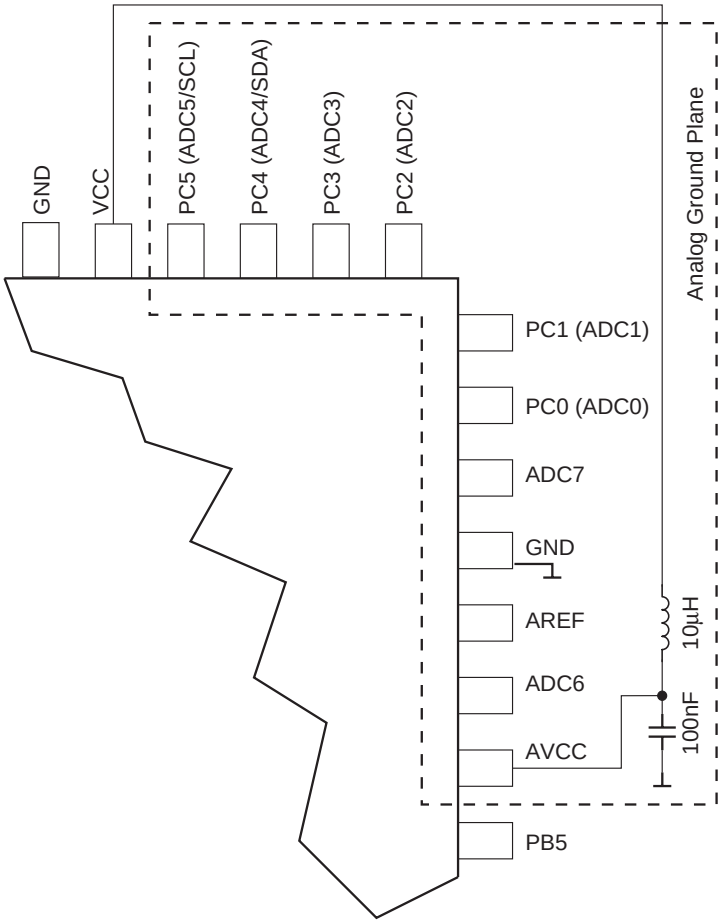


模拟噪声抑制技术

设备内部及外部的数字电路都会产生电磁干扰 (EMI)，从而影响模拟测量的精度。如果转换精度要求较高，那么可以通过以下方法来减少噪声：

1. 模拟通路越短越好。保证模拟信号线位于模拟地之上，并使它们与高速切换的数字信号线分开。
2. 如 Figure109 所示，AVCC 应通过一个 LC 网络与数字电压源 V_{CC} 连接。
3. 使用 ADC 噪声抑制器来降低来自 CPU 的干扰噪声。
4. 如果有 ADC[3..0] 端口被用作数字输出，那么必须保证在转换进行过程中它们不会有电平的切换。但 TWI 接口 (ADC4 和 ADC5) 只影响 ADC4 与 ADC5，而不会影响其他 ADC 通道。

Figure 109. ADC 电源连接图



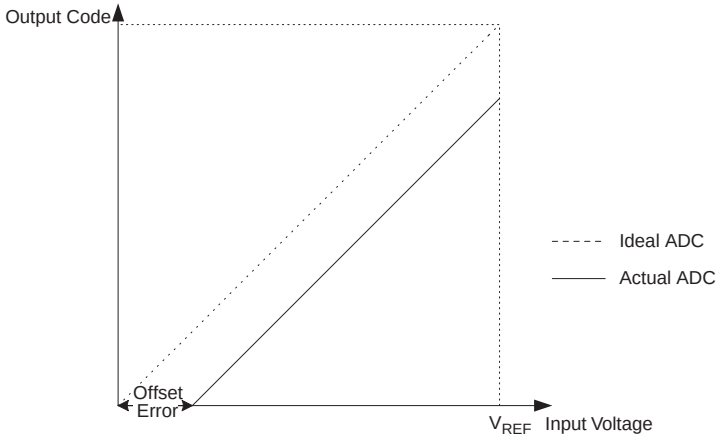
ADC 精度定义

一个 n 位的单端 ADC 将 GND 与 V_{REF} 之间的线性电压转换成 2^n 个 (LSBs) 不同的数字量。最小的转换码为 0，最大的转换码为 2^n-1。

以下几个参数描述了与理想情况之间的偏差：

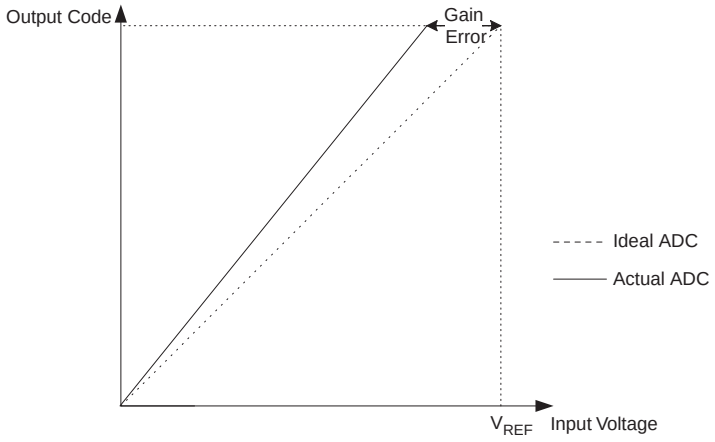
- 偏移：第一次转换 (0x000 到 0x001) 与理想转换 (0.5 LSB) 之间的偏差。理想情况：0 LSB。

Figure 110. 偏移误差



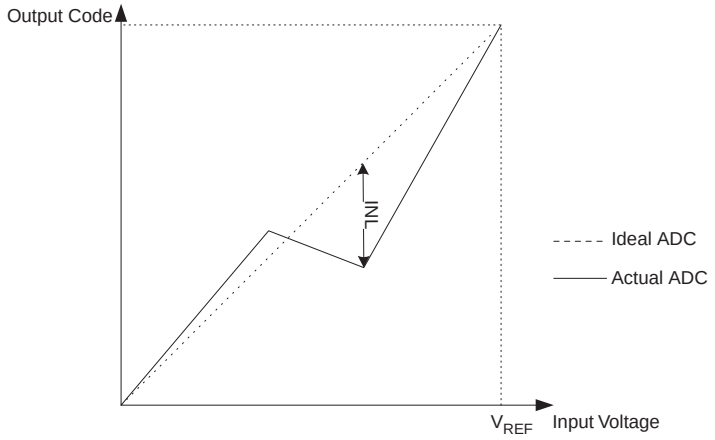
- 增益误差 : 调整偏差之后 , 最后一次转换 (0x3FE 到 0x3FF) 与理想情况 (最大值以下 1.5 LSB) 之间的偏差即为增益误差。理想值为 0 LSB。

Figure 111. 增益误差



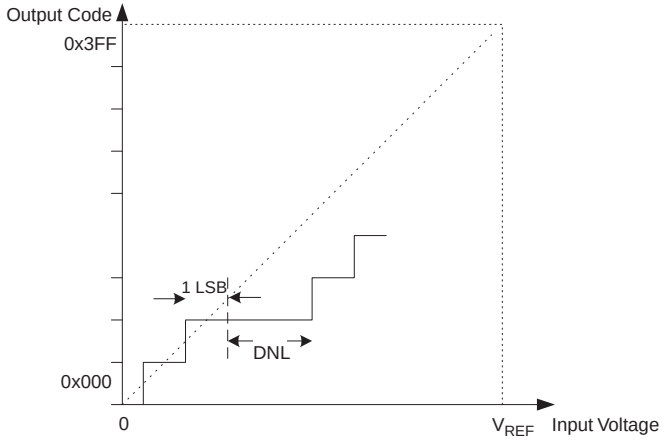
- 整体非线性 (INL) : 调整偏移及增益误差之后 , 所有实际转换与理想转换之间的最大误差即为 INL。理想值 : 0 LSB。

Figure 112. 整体非线性 (INL)



- 差分非线性(DNL):实际码宽(两个邻近转换之间的码间距)与理论码宽(1 LSB)之间的偏差。理论值 : 0 LSB。

Figure 113. 差分非线性 (DNL)



- 量化误差：由于输入电压被量化成有限位的数码，某个范围的输入电压 (1 LSB) 被转换为相同的数码。量化误差总是为 ± 0.5 LSB。
- 绝对精度：所有实际转换 (未经调整) 与理论转换之间的最大偏差。由偏移、增益误差、差分误差、非线性及量化误差构成。理想值为 ± 0.5 LSB。

ADC 转换结果

转换结束后 (ADIF 为高)，转换结果被存入 ADC 结果寄存器 (ADCL, ADCH)。

单次转换的结果如下：

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

式中，V_{IN} 为被选中引脚的输入电压，V_{REF} 为参考电压 (参见 P226Table99 与 P227Table100). 0x000 代表模拟地电平，0x3FF 代表所选参考电压的数值减去 1LSB。

ADC 多路复用选择寄存器— ADMUX

位	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
读 / 写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7:6 – REFS1:0: 参考电压选择**

如 Table99 所示，通过这几位可以选择参考电压。如果在转换过程中改变了它们的设置，只有等到当前转换结束 (ADCSRA 寄存器的 ADIF 置位) 之后改变才会起作用。如果在 AREF 引脚上施加了外部参考电压，内部参考电压就不能被选用了。

Table 99. ADC 参考电压选择

REFS1	REFS0	参考电压选择
0	0	AREF，内部 Vref 关闭
0	1	AVCC，AREF 引脚外加滤波电容
1	0	保留
1	1	1.1V 的片内基准电压源，AREF 引脚外加滤波电容

- **位 5 – ADLAR: ADC 转换结果 左对齐**

ADLAR影响ADC转换结果在ADC数据寄存器中的存放形式。ADLAR置位时转换结果为左对齐，否则为右对齐。 ADLAR 的改变将立即影响 ADC 数据寄存器的内容，不论是否有转换正在进行。关于这一位的完整描述请见 P229”ADC 数据寄存器 – ADCL 及 ADCH” 。

- **位 4 – Res: 保留位**

ATmega48/88/168 中该位未使用，且读返回值为 "0”。

- **位 3:0 – MUX3:0: 模拟通道选择位**

通过这几位的设置，可以对连接到 ADC 的模拟输入进行选择。细节见 Table100。如果在转换过程中改变这几位的值，那么只有到转换结束 (ADCSRA 寄存器的 ADIF 置位) 后新的设置才有效。

Table 100. 输入通道选择

MUX3..0	单端输入
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	(保留)
1001	(保留)
1010	(保留)
1011	(保留)
1100	(保留)
1101	(保留)
1110	1.1V (V_{BG})
1111	0V (GND)

ADC 控制及状态寄存器 A— ADCSRA

位	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – ADEN: ADC 使能**

ADEN置位即启动ADC ,否则ADC功能关闭。在转换过程中关闭ADC将立即中止正在进行的转换。

- **位 6 – ADSC: ADC 开始转换**

在单次转换模式下，ADSC 置位将启动一次 ADC 转换。在连续转换模式下，ADSC 置位将启动首次转换。第一次转换 (在 ADC 启动之后置位 ADSC , 或者在使能 ADC 的同时置位 ADSC) 需要 25 个 ADC 时钟周期，而不是正常情况下的 13 个。第一次转换执行 ADC 初始化的工作。

在转换进行过程中读取 ADSC 的返回值为 "1”，直到转换结束。ADSC 清零不产生任何动作。

- **位 5 – ADATE: ADC 自动触发使能**

ADATE 置位将启动 ADC 自动触发功能。触发信号的上跳沿启动 ADC 转换。触发信号源通过 ADCSRB 寄存器的 ADC 触发信号源选择位 ADTS 设置。

- **位 4 – ADIF: ADC 中断标志**

在 ADC 转换结束，且数据寄存器被更新后， ADIF 置位。如果 ADIE 及 SREG 中的全局中断使能位 I 也置位，ADC 转换结束中断服务程序即得以执行，同时 ADIF 硬件清零。此外，还可以通过向此标志写 1 来清 ADIF。要注意的是，如果对 ADCSRA 进行读 - 修改 - 写操作，那么待处理的中断会被禁止。这也适用于 SBI 及 CBI 指令。

- **位 3 – ADIE: ADC 中断使能**

若 ADIE 及 SREG 的位 I 置位， ADC 转换结束中断即被激活。

- **位 2:0 – ADPS2:0: ADC 预分频器选择位**

这几位确定了 XTAL 与 ADC 输入时钟之间的分频因子。

Table 101. ADC 预分频选择

ADPS2	ADPS1	ADPS0	分频因子
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADC 数据寄存器—ADCL 及 ADCH

ADLAR = 0

位	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

位	15	14	13	12	11	10	9	8	
	ADC9	**ADC8**	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADCH**
	ADC1	**ADC0**	**–**	**–**	**–**	**–**	**–**	**–**	**ADCL**
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADC 转换结束后，转换结果存于这两个寄存器之中。

读取 ADCL 之后，ADC 数据寄存器一直要等到 ADCH 也被读出才可以进行数据更新。因此，如果转换结果为左对齐，且要求的精度不高于 8 比特，那么仅需读取 ADCH 就足够了。否则必须先读出 ADCL 再读 ADCH。

ADMUX 寄存器的 ADLAR 及 MUXn 会影响转换结果在数据寄存器中的表示方式。如果 ADLAR 为 1，那么结果为左对齐；反之 (系统缺省设置)，结果为右对齐。

- **ADC9:0: ADC 转换结果**

ADC 转换的结果，细节见 P227”ADC 转换结果 ” 。

ADC 控制及状态寄存器 B—ADCSRB

位	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
读 / 写	R	R/W	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7, 5:3 – Res: 保留位**

这几位保留。为了与以后的器件相兼容，在写 ADCSRB 时这几位应写 "0”。

- **位 2:0 – ADTS2:0: ADC 自动触发源**

若 ADCSRA 寄存器的 ADATE 置位，ADTS 的值将确定触发 ADC 转换的触发源；否则，ADTS 的设置没有意义。被选中的中断标志在其上升沿触发 ADC 转换。从一个中断标志清零的触发源切换到中断标志置位的触发源会使触发信号产生一个上升沿。如果此时 ADCSRA 寄存器的 ADEN 为 1 ,ADC 转换即被启动。切换到连续运行模式 (ADTS[2:0]=0) 时，即使 ADC 中断标志已经置位也不会产生触发事件。

Table 102. ADC 自动触发源选择

ADTS2	ADTS1	ADTS0	触发源
0	0	0	连续转换模式
0	0	1	模拟比较器
0	1	0	外部中断请求 0
0	1	1	定时器 / 计数器 0 比较匹配
1	0	0	定时器 / 计数器 0 溢出
1	0	1	定时器 / 计数器比较匹配 B
1	1	0	定时器 / 计数器 1 溢出
1	1	1	定时器 / 计数器 1 捕捉事件

数字输入禁止寄存器 0—DIDR0

位	7	6	5	4	3	2	1	0	
	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
e 初始值	0	0	0	0	0	0	0	0	

- **位 7:6 – Res: 保留位**

这几位保留。为了与以后的器件相兼容，在写 DIDR0 时这几位应写 "0”。

- **位 5..0 – ADC5D..ADC0D: ADC5..0 数字输入禁止**

如果这几位为 "1”，那么对应 ADC 引脚的数字输入缓冲器被禁止， PIN 寄存器的对应位将为 "0”。如果 ADC5..0 引脚施加了模拟信号，且当前应用不需要这些管脚提供数字输入缓冲器时，应向这几位写 "1” 来降低数字输入缓冲器的功耗。

注意 ADC 的引脚 ADC7 与 ADC6 没有数字输入缓冲器，且不需要数字输入禁止位。

片上调试系统

特性

- 完全的程序流控制
- 仿真芯片上所有的模拟和数字功能，除了 RESET 引脚
- 实时操作
- 支持符号调试 (C 与汇编级，或其它 HLL)
- 没有限制的程序断点数 (使用软件断点)
- 非插入式操作
- 与实际器件相同的电气特性
- 自动配置系统
- 高速操作
- 编程非易失性存储器

概述

debugWIRE 片上调试系统使用单线双向接口来控制程序流，在 CPU 中执行 AVR 指令，对不同的非易失性存储器进行编程。

物理接口

当 debugWIRE 使能熔丝位 DWEN 被编程且锁定位未编程时，目标器件中的 debugWIRE 系统被激活。RESET 端口引脚配置为上拉使能的线与 (开漏) 双向 I/O，成为目标与仿真器间的联系通路。

Figure 114. debugWIRE 设置

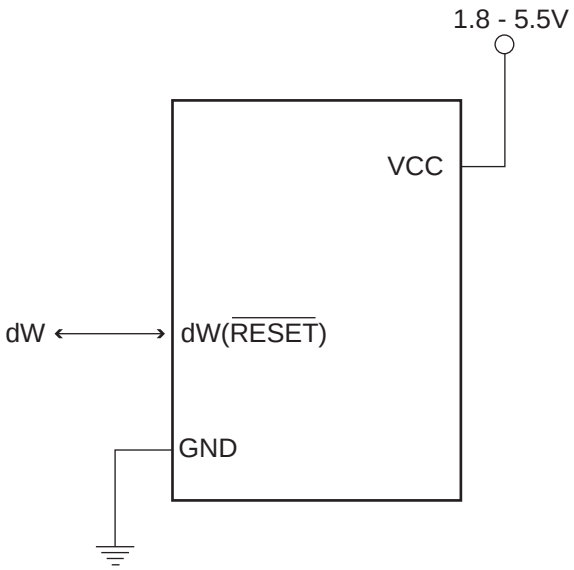


Figure114 给出 debugWIRE 使能的目标 MCU 及仿真连接器的示意图。系统时钟不受 debugWIRE 的影响，只由 CKSEL 熔丝位决定。

设计使用 debugWIRE 的系统时，必须进行下面的检查：

- dW/(RESET) 的上拉电阻不得小于 10kΩ。 debugWIRE 并不需要上拉电阻
- 将 RESET 引脚与 V_{CC} 直接连接将无法工作
- 使用 debugWIRE 时必须断开与 RESET 引脚连接的电容
- 必须断开所有的外部复位源

软件断点

debugWIRE 通过 AVR 断点指令来设置程序存储器断点。在 AVR Studio® 设置一个断点将在程序存储器中插入 BREAK 指令。被 BREAK 指令所替代的指令将被保存。程序继续运

行时，保存的指令得到执行，然后继续执行其他指令。断点也可以通过在程序中插入 BREAK 指令进行手工设置。

每次断点改变后 Flash 必须要重新编程。这由 AVR Studio® 通过 debugWIRE 接口自动处理。断点的使用会降低 Flash 数据记忆时间。调试用的器件不能发给最终客户。

debugWIRE 的局限

debugWIRE 通讯引脚 (dW) 与外部复位 (RESET) 共用同一引脚。因此使能 debugWIRE 之后，系统不支持外部复位源。

当程序在 CPU 中全速运行时， debugWIRE 系统精确的仿真所有的 I/O 口功能；当 CPU 停止工作时，通过调试器访问某些 I/O 寄存器时要注意。详见 debugWIRE 文档。

DWEN 熔丝位的编程使部分时钟系统在所有的休眠模式下都保持运行。这会增加器件休眠模式的功耗。因此不使用 debugWire 时应该禁用 DWEN 熔丝位。

I/O 存储器中与 debugWIRE 相关的寄存器

下面说明在 debugWire 中用到的寄存器。

debugWire 数据寄存器—DWDR

位	7	6	5	4	3	2	1	0	
	DWDR[7:0]								DWDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

DWDR 寄存器为在 MCU 中运行的程序与调试器提供了通信通路。该寄存器只能由 debugWIRE 访问且不能在通常操作中作为通用寄存器使用。

Flash 自编程，ATmega48

ATmega48 不支持同时读写 (R-W-W)，且没有独立的 Boot Loader 区。SPM 指令可以在整个 Flash 区中运行。

器件为通过 MCU 本身来下载和上载程序代码提供了一个自编程机制。自编程可以使用任何器件可用的数据接口和相关的协议来获得代码并把代码 (程序) 写入程序存储器。

程序存储器的更新以页的方式进行。在用临时页缓冲器存储的数据对一页存储器进行编程之前首先要将这一页擦除。 SPM 指令以一次一个字的方式将数据写入临时页缓冲器。临时页缓冲器的写入可以在页擦除命令之前完成，也可以在页擦除和页写操作之间完成。

方案 1，在页擦除前填充缓冲器

- 填充临时页缓冲器
- 执行页擦除操作
- 执行页写操作

方案 2，在页擦除之后填充缓冲器

- 执行页擦除操作
- 填充临时页缓冲器
- 执行页写操作

如果只需要改变页中的一部分，擦除前必须保存页中的其它部分 (例如保存于临时页缓冲器之中)，再重新写入。使用方案 1 时， Boot Loader 提供了有效的读 - 该 - 写操作，允许用户先读页，做必要的改变，再写回修改后的数据；若使用方案 2，由于页已被擦除，因此不可能在加载数据时读取旧的数据。临时页缓冲器可以进行随机访问。进行页擦除与页写操作时要确保地址是相同的。

通过 SPM 完成页擦除

执行页擦除操作首先需要设置 Z 指针的地址信息，然后将 “00000011” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 在擦除操作过程中 CPU 停止

写临时缓冲区 (页加载)

写一个指令字首先需要设置 Z 指针的地址信息，以及将指令字写入 R1:R0，然后将 “00000001” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。 Z 寄存器中 PCWORD 的内容用来寻址临时缓冲区。页写操作完成，或置位 SPMCSR 寄存器的 RWWSRE 将使临时缓冲区自动擦除。系统复位也会擦除临时缓冲区。但是如果不清除临时缓冲区就只能对每个地址进行一次写操作。

如果在 SPM 页加载操作过程中对 EEPROM 执行了写操作，则所有加载的数据都将丢失。

执行页写操作

执行页写操作首先需要设置 Z 指针的地址信息，然后将 “00000101” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 页写过程中 CPU 停止

在自编程过程中寻址 Flash

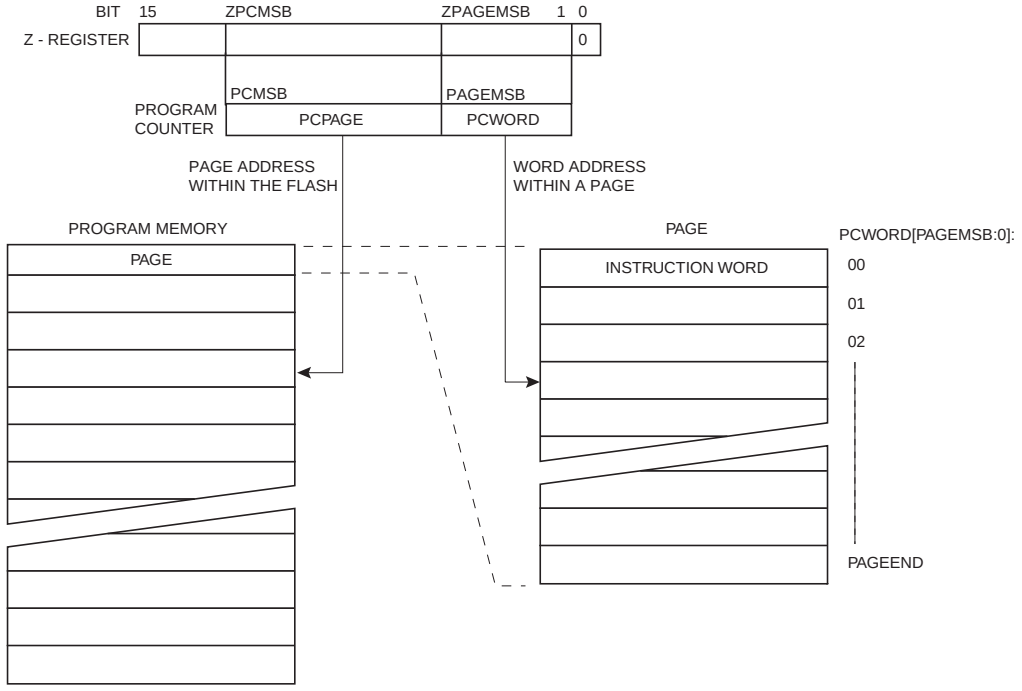
Z 指针用来寻址 SPM 命令。

位	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

由于 Flash 存储器是以页的形式组织 (P261Table126) 起来的，程序计数器可看作由两个部分构成：其一为实现页内寻址的低位部分；其次为实现页寻址的高位部分，如 Figure118所示。由于页擦除和页写操作的寻址是相互独立的，因此保证 Boot Loader 软件在页擦除和页写操作时寻址相同的页是最重要的。

LPM指令也使用Z指针来保存地址。由于这个指令的寻址逐字节地进行 ,所以Z指针的LSB位 (位 Z0) 也使用到了。

Figure 115. SPM(1) 的寻址



Note: 1. Figure118 中所用的不同的变量在 P261Table126 列出。

存贮程序存储器 (SPM) 控制和状态寄存器—SPMCSR

SPMCSR 包括了控制 Boot Loader 操作所需的控制位。

位	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – SPMIE: SPM 中断使能**

SPMIE 置位后，如果状态寄存器的 I 位也置位，SPM 中断即被使能。只要 SPMCSR 寄存器的 SPMEN 清零， SPM 中断将被执行。

- **位 6 – RWWSB:RWW 区忙标志**

RWWSB 用来兼容支持 RWW 的器件，在 ATmega48 中读返回值为 "0”。

- **位 5 – Res: 保留位**

在 ATmega48/88/168 中为保留位，读返回值为 "0”。

- **位 4 – RWWSRE: RWW 区读使能**

该位在 ATmega48 中的功能是其在 ATmega88/168 中的一部分。如果临时页缓冲器填充与 RWWSRE 写操作同时发生，则临时页缓冲器清零，加载的数据亦将丢失。

- **位 3 – BLBSET: Boot 锁定位设置**

该位在 ATmega48 中的功能是其在 ATmega88/168 中的一部分。在 SPMCSR 的 BLBSET 位和 SPMEN 位 置位后的三个周期内运行的 LPM 指令将读取锁定位或熔丝位 (取决于 Z 指针的 Z0) 并送到目的寄存器。详见 P237” 通过软件读取熔丝位和锁定位 ” 。

- **位 2 – PGWRT: 页写入**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页写功能，将临时缓冲器中存储的数据写入 Flash。页地址取自 Z 指针的高位部分。R1 和 R0 的数据则被忽略。页写操作完成，或在四个时钟周期内没有 SPM 指令被执行时，PGWRT 自动清零。如果页写对象为 NRWW 区，在整个页写操作过程中 CPU 停止。

- **位 1 – PGERS: 页擦除位**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页擦除功能。页地址取自 Z 指针的高位部分。 R1 和 R0 的数据则被忽略。页擦除操作完成，或在四个时钟周期内没有SPM指令被执行时，PGERS自动清零。如果页写对象为NRWW区，在整个页擦除操作过程中 CPU 停止。

- **位 0 – SPMEN: 存贮程序存储器使能位**

这一位使能紧接着的四个时钟周期内的 SPM 指令。如果将这一位和 RWWSRE、BLBSET、PGWRT 或 PGERS 之一同时置位，则如上所述，接下来的 SPM 指令将有特殊的含义。如果只有 SPMEN 置位，那么接下来的 SPM 指令将把 R1:R0 中的数据存储到由 Z 指针确定的临时页缓冲器。Z 指针的 LSB 被忽略。SPM 指令完成，或在四个时钟周期内没有 SPM 指令被执行时， SPMEN 自动清零。在页擦除和页写过程中 SPMEN 保持为高直到操作完成。

在低五位中写入除 “10001”、“01001”、“00101”、 “00011” 或 “00001” 之外的任何组合都无效。

EEPROM 写操作阻止对 SPMCSR 寄存器的写操作

EEPROM 写操作会阻塞对 Flash 的编程，也会阻塞对熔丝位和锁定位的读操作。建议用户在对 SPMCSR 寄存器进行写操作之前首先检查 EECR 寄存器的状态位 EEWE，确保此位已被清除。

通过软件读取熔丝位和锁定位

熔丝位和锁定位可以通过软件读取。读锁定位时，需要将 0x0001 赋予给 Z 指针并且置位 SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的 LPM 指令将把锁定位的值将加载到目的寄存器。读锁定位操作结束，或者在三个 CPU 周期内没有 执行LPM指令，或在四个CPU周期内没有执行SPM指令，BLBSET 和 SPMEN 位将自动硬件清零。 BLBSET 和 SPMEN 清零后，LPM 将按照指令手册中所描述的那样工作。

位	7	6	5	4	3	2	1	0
Rd	–	–	–	–	–	–	LB2	LB1

读取熔丝位低字节的算法和上述读取锁定位的算法类似。要读取熔丝位低字节，需要将 0x0000赋予给Z指针并且置位SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位低位字节的值 (FLB) 加载到目的寄存器。更详细的说明及熔丝位低位字节映射的细节请参见 P256Table119 。

位	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

类似的，读取熔丝位高位字节时，需要将 0x0003 赋予给 Z 指针并且置位 SPMCSR 寄存器的 BLBSET 和 SPMEN。在 SPMCSR 操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位高位字节的值 (FHB) 加载到目的寄存器。更详细的说明及熔丝位高位字节映射的细节请参见 P256Table118 。在 SPMCSR 操作之后的三个 CPU 周期内执行的 LPM 指

位	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

令将把熔丝位扩展字节的值 (EFB) 加载到目的寄存器。更详细的说明及熔丝位扩展字节映射的细节请参见 P256Table119 。

位	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

被编程的熔丝位 / 锁定位的读返回值为 "0”。未被编程的熔丝位 / 锁定位的读返回值为 "1”。

防止 Flash 损毁

V_{CC} 低于工作电压时，CPU 和 Flash 正常工作无法保证，Flash 的内容可能受到破坏。这个问题在板级系统的独立 Flash 中一样存在。所以也要采用同样的解决方案。

电压太低时有两种情况可以破坏 Flash 内容。第一，Flash 写过程需要一个最低电压。第二，电压太低时 CPU 本身会错误地执行指令。

通过遵循以下设计建议可以避免 Flash 被破坏 (采用其中之一就足够了) :

1. 电源电压不足期间，保持 AVR RESET 为低：采用的方式为：如果工作电压与检测电平相匹配，可以使能 BOD 功能；否则可以使用外部复位保护电路。如果在写操作进行中发生了复位，只要电源电压足够，写操作还会完成。
2. 低电压期间保持 AVR 内核处于掉电休眠模式。这样可以防止 CPU 解码并执行指令，有效地保护 SPMCSR 寄存器，从而保护 Flash 被无意识得修改掉。

使用 SPM 时的 Flash 编程时间

片内校准的 RC 振荡器用于 Flash 寻址时序控制。Table108 给出了 CPU 访问 Flash 的典型编程时间。

Table 103. SPM 编程时间

符号	最小编程时间	最大编程时间
Flash 写操作 (通过 SPM 实现页擦除、页写、及写锁定位)	3.7 ms	4.5 ms

简单的 Boot Loader 汇编代码例子

注意在 ATmega48 中 RWWSB 位的读返回值为 "0”。但是仍然建议如代码所示那样检查这一位位，以保证与支持 RWW 的器件相兼容。

```
  ;- 本例程将 RAM 中的一页数据写入 Flash
  ; Y 指针指向 RAM 的第一个数据单元
  ;Z 指针指向 Flash 的第一个数据单元
  ;- 本例程没有包括错误处理
  ;- 该程序必须放置于 Boot 区 ( 至少 Do_spm 子程序是如此 )
  ; 在自编程过程中 ( 页擦除和页写操作 ) 只能读访问 NRWW 区的代码
  ;- 使用的寄存器：r0、 r1、 temp1 (r16)、 temp2 (r17)、 looplo (r24) 、
  ; loophi (r25)、 spmcrval (r20)
  ; 在程序中不包括寄存器内容的保护和恢复
  ; 在牺牲代码大小的情况下可以优化寄存器的使用
  ;- 假设中断向量表位于 Boot loader 区，或者中断被禁止。
.equ     PAGESIZEB = PAGESIZE*2                  ;PAGESIZEB 是以字节为单位的页
大小，不是以字为单位
.org SMALLBOOTSTART
Write_page:
  ;   页擦除
  ldi         spmcrval, (1<<PGERS) | (1<<SPMEN)
  rcall       Do_spm

  ;   重新使能 RWW 区
  ldi         spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  rcall       Do_spm

  ;   将数据从 RAM 转移到 Flash 页缓冲区
  ldi         looplo, low(PAGESIZEB)              ; 初始化循环变量
  ldi         loophi, high(PAGESIZEB)             ;PAGESIZEB<=256 时不需要此操
作
Wrloop:
  ld          r0, Y+
  ld          r1, Y+
  ldi         spmcrval, (1<<SPMEN)
  rcall       Do_spm
  adiw        ZH:ZL, 2
  sbiw        loophi:looplo, 2                    ;PAGESIZEB<=256时请使用subi
  brne        Wrloop

  ;   执行页写操作
  subi        ZL, low(PAGESIZEB)                  ; 复位指针
  sbci        ZH, high(PAGESIZEB)                 ;PAGESIZEB<=256 时不需要此操
作
  ldi         spmcrval, (1<<PGWRT) | (1<<SPMEN)
  rcall       Do_spm

  ;   重新使能 RWW 区
  ldi         spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  rcall       Do_spm

  ;   读回数据并检查，为可选操作
  ldi         looplo, low(PAGESIZEB)              ; 初始化循环变量
  ldi         loophi, high(PAGESIZEB)             ;PAGESIZEB<=256 时不需要此操
作
```

```
  subi        YL, low(PAGESIZEB)                    ;复位指针
  sbci        YH, high(PAGESIZEB)
Rdloop:
  lpm         r0, Z+
  ld          r1, Y+
  cpse        r0, r1
  rjmp        Error
  sbiw        loophi:looplo, 1                      ;PAGESIZEB<=256 时请使用 subi
  brne        Rdloop

  ;   返回到 RWW 区
  ;   确保 RWW 区已经可以安全读取
Return:
  in          temp1, SPMCSR
  sbrs        temp1, RWWSB                          ; 若RWWSB为"1",说明RWW区还没有准备
好
  ret
  ;   重新使能 RWW 区
  ldi         spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  rcall       Do_spm
  rjmp        Return

Do_spm:
  ;   检查先前的 SPM 操作是否已经完成
Wait_spm:
  in          temp1, SPMCSR
  sbrc        temp1, SPMEN
  rjmp        Wait_spm
  ;       输入：spmcrval 决定了 SPM 操作
  ;   禁止中断，保存状态标志
  in          temp2, SREG
  cli
  ;   确保没有 EEPROM 写操作
Wait_ee:
  sbic        EECR, EEPE
  rjmp        Wait_ee
  ;   SPM 时间序列
  out         SPMCSR, spmcrval
  spm
  ;   恢复 SREG ( 如果中断原本是使能的，则使能中断)
  out         SREG, temp2
  ret
```

Boot Loader 支持 RWW 自编程，ATmega88 与 ATmega168

在 ATmega88 与 ATmega168 中，Boot Loader 为通过 MCU 本身来下载和上载程序代码提供了一个真正的同时读 - 写 (Read-While-Write，以下简称 RWW) 自编程机制。这一特点使得系统可以在 MCU 的控制下，通过驻留于程序 Flash 的 Boot Loader，灵活地进行应用软件升级。Boot Loader 可以使用任何器件具有的数据接口和相关的协议获得代码并把代码（程序）写入 Flash，或者从程序存储器读取代码。 Boot Loader 区的程序可以写整个 Flash，包括 Boot Loader 区本身。因而 Boot Loader 可以对其自身进行修改，甚至将自己擦除。 Boot Loader 存储器空间的大小可以通过熔丝位进行配置。Boot Loader 具有两套程序加密位，各自可以独立设置，给用户提供了选择保护级的灵活性。

Boot Loader 的特点

- **RWW 自编程**
- **灵活的 Boot Loader 存储区配置**
- **高度的安全性 (有单独的 Boot 锁定位实现灵活的程序保护)**
- **有独立的熔丝位用于选择复位向量**
- **优化的页 (1) 大小**
- **代码优化的算法**
- **有效的 RWW 支持**

Note: 1. 页是Flash的一个部分，由数个字节组成(见P261Table126)，在编程过程中使用。页的组织结构不影响正常的操作。

应用 Flash 区及 Boot Loader Flash 区

Flash 由两个区构成，应用区和 Boot Loader 区 (见 Figure117)。两个区的存储空间大小由 BOOTSZ 熔丝位配置，如 P252Table109 和 Figure117 所示。由于两个区使用不同的锁定位，所以可以具有不同的加密级别。

应用区

应用区是 Flash 用来存储应用代码的区域。应用区的保护级别通过应用 Boot 锁定位 (Boot 锁定位 0) 确定，详见 P243Table105 。由于 SPM 指令在应用区执行时是无效的，所以应用区不能用来存储 Boot Loader 代码。

BLS—Boot Loader 区

应用区用来存储应用代码，而 Boot Loader 软件必须保存在 BLS。这是因为只有在 BLS 运行时 SPM 指令才有效。SPM 指令可以访问整个 Flash，包括 BLS 本身。Boot Loader 区的保护级别通过 Boot Loader 锁定位 (Boot 锁定位 1) 确定，详见 P243Table106 。

RWW 和非 RWW Flash 区

CPU 是否支持 RWW，或者 CPU 是否在利用 Boot Loader 软件进行代码更新时停止，取决于被编程的是哪个地址。除了前面所述的通过 BOOTSZ 熔丝位配置的两个区之外，Flash 还可以分成两个固定的区——同时读 - 写 (RWW) 区和非同时读 - 写 (NRWW) 区。RWW- 和 NRWW 的分界在 P252Table110 和 P242Figure 117 给出。两个区的主要区别是：

- 对 RWW 区内的页进行擦除或写操作时可以读 NRWW 区
- 对 NRWW 区内的页进行擦除或写操作时， CPU 停止

注意，Boot Loader 软件工作时，用户软件不能读取位于 RWW 区内的任何代码。"RWW 区“ 指的是被编程 (擦除或写) 的那个存储区，而不是利用 Boot Loader 软件进行代码更新过程中实际被读取的那部分。

RWW—同时读—写存储区

如果 Boot Loader 软件是对 RWW 区内的某一页进行编程，则可以从 Flash 中读取代码，但只限于 NRWW 区内的代码。在 Flash 编程期间，用户软件必须保证没有对 RWW 区的读访问。如果用户软件在编程过程中试图读取位于 RWW 区的代码 (如通过 call/jmp/lpm 指令或中断)，软件可能会终止于一个未知状态。为了避免这种情况的发生，需要禁止中断或将其转移到 Boot Loader 区。Boot Loader 总是位于 NRWW 存储区。只要 RWW 区处于不能读访问的状态，存储程序存储器控制和状态寄存器 (SPMCSR) 的 RWW 区忙标志位 RWWSB 置位。编程结束后，要在读取 位于 RWW 区的代码之前通过软件清除 RWWSB。具体如何清除 RWWSB 请参见 P244“ 存贮程序存储控制器和状态寄存器 – SPMCSR” 。

NRWW—非同时读—写存储区

在 Boot Loader 软件更新 RWW 区的某一页时，可以读取位于 NRWW 区的代码。当 Boot Loader 代码更新 NRWW 区时，在整个页擦除或写操作过程中 CPU 被挂起。

Table 104. RWW 的特点

编程过程中 Z 指针寻址哪个区？	编程过程中可以读取哪个区？	CPU 挂起吗？	支持 RWW 吗？
RWW 区	NRWW 区	不	是
NRWW 区	无	是	不

Figure 116. RWW 区与 NRWW 区

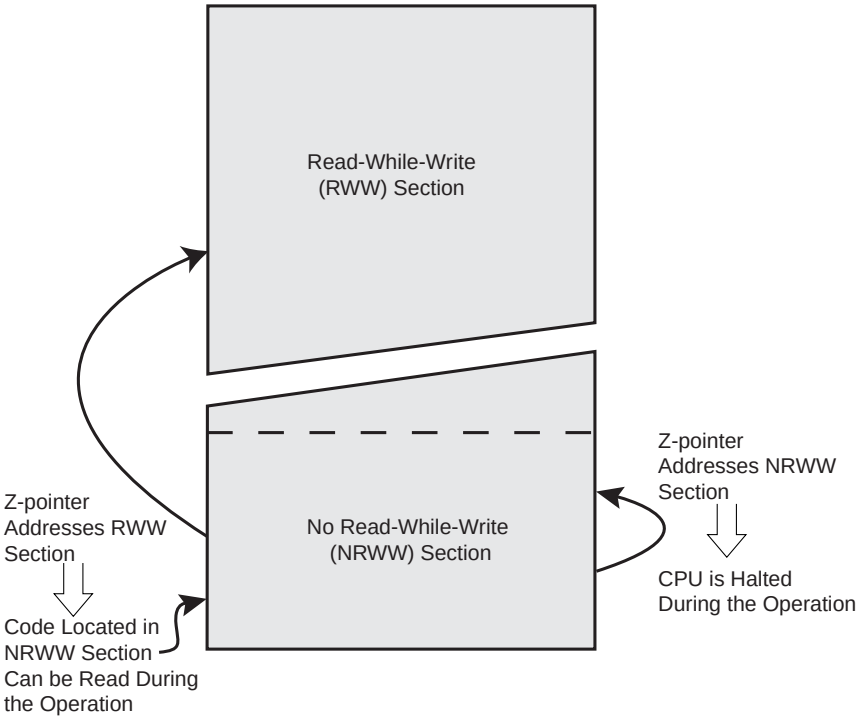
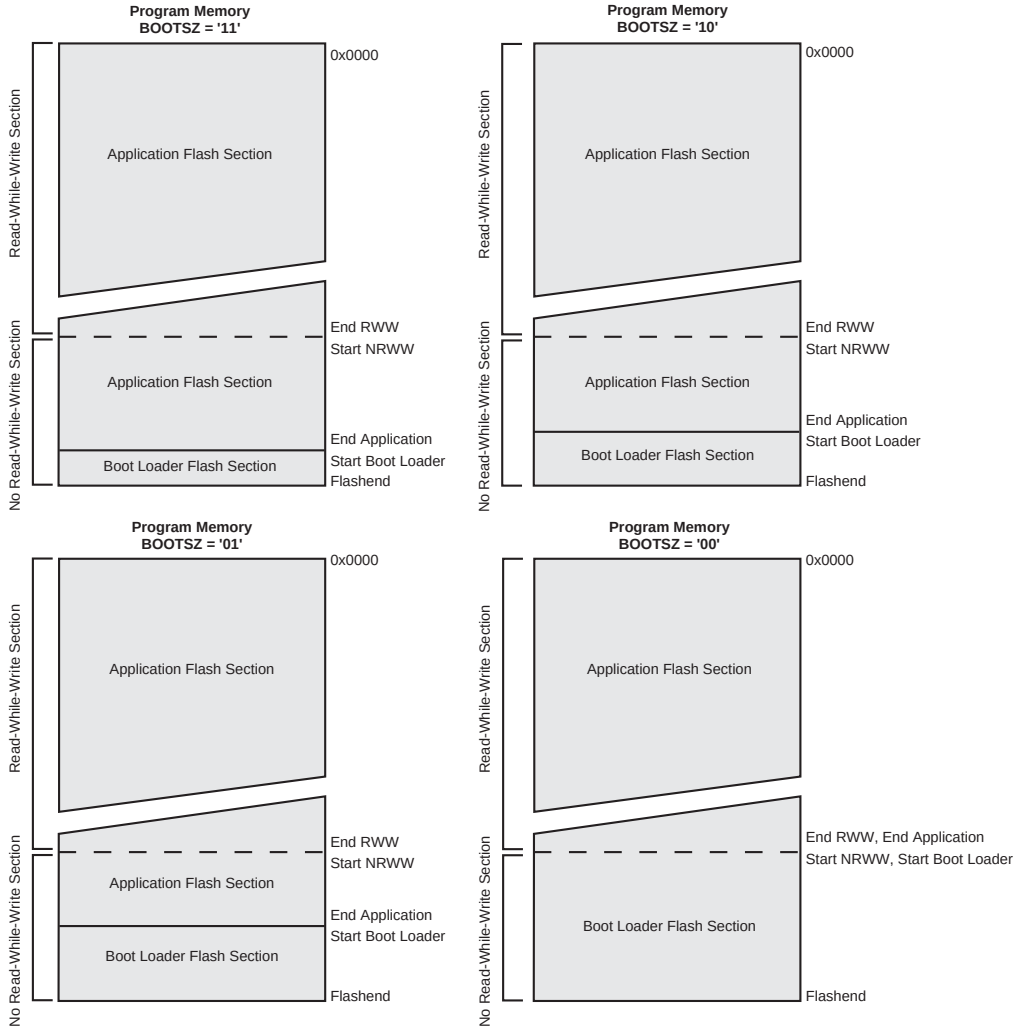


Figure 117. 存储区



Note: 1. 上图中的参数在 P252Table109 中给出。

Boot Loader 锁定位

如果不需要 Boot Loader 功能，则整个 Flash 都可以为应用代码所用。Boot Loader 具有两套可以独立设置的 Boot 锁定位。用户可以灵活地选择不同的代码保护方式。

用户可以选择：

- 保护整个 Flash 区，不让 MCU 进行软件升级
- 不允许 MCU 升级 Boot Loader Flash 区
- 不允许 MCU 升级应用 Flash 区
- 允许 MCU 升级整个 Flash 区

详细内容请参见 Table105 和 Table106。 Boot 锁定位可以通过软件、串行下载或并行编程进行设置，但只能通过芯片擦除命令清除。通用的写锁定位 (锁定位模式 2) 不限制通过 SPM 指令对 Flash 进行编程。与此类似，通用的读 / 写锁定位 (锁定位模式 1) 也不限制通过 LPM/SPM 指令对闪存进行读 / 写访问。

Table 105. Boot 锁定位 0 保护模式 (应用区)(1)

BLB0 模式	BLB02	BLB01	保护
1	1	1	允许 SPM/LPM 指令访问应用区
2	1	0	不允许 SPM 指令对应用区进行写操作
3	0	0	不允许 SPM 指令对应用区进行写操作，也不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。

Note: 1. “1” 表示未被编程，“0” 表示已编程。

Table 106. Boot 锁定位 1 保护模式 (Boot Loader 区)(1)

BLB1 模式	BLB12	BLB11	保护
1	1	1	允许 SPM/LPM 指令访问 Boot Loader 区
2	1	0	不允许 SPM 指令对 Boot Loader 区进行写操作
3	0	0	不允许 SPM 指令对 Boot Loader 区进行写操作，也不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。

Note: 1. “1” 表示未被编程，“0” 表示已编程。

进入 Boot Loader 程序

通过跳转指令或从应用区调用的方式可以进入 Boot Loader。这些操作可以由一些触发信号启动，比如通过 USART 或 SPI 接口接收到了相关的命令。另外，可以通过编程 Boot 复位熔丝位使得复位向量指向 Boot 区的起始地址。这样，复位后 Boot Loader 立即就启动了。加载了应用代码后，程序开始执行应用代码。 MCU 本身不能改变熔丝位的设置。也就是说，一旦 Boot 复位熔丝位被编程，复位向量将一直指向 Boot 区的起始地址。熔丝位只能通过串行或并行编程的方法来改变。

Table 107. Boot 复位熔丝位 (1)

BOOTRST	复位地址
1	复位向量 = 应用区复位 (地址 0x0000)
0	复位向量 =Boot Loader 复位 (见 P252Table109)

Note: 1. “1” 意味着未编程，“0” 意味着已编程。

存贮程序存储控制器和状态寄存器—SPMCSR

存贮程序存储控制器和状态寄存器包括了控制 Boot Loader 操作所需的控制位。

位	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
读 / 写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – SPMIE: SPM 中断使能**

SPMIE 置位后，如果状态寄存器的 I 位也置位，SPM 中断即被使能。只要 SPMCSR 寄存器的 SPMEN 清零， SPM 中断将被执行。

- **位 6 – RWWSB:RWW 区忙标志**

启动对 RWW 区的自编程（页擦除或页写入）操作时， RWWSB 被硬件置 1。 RWWSB 置位时不能访问 RWW 区。自编程操作完成后，如果 RWWSRE 位为 1， RWWSB 位将被清除。另外，启动页加载操作将使 RWWSB 位自动清零。

- **位 5 – Res: 保留位**

在 ATmega48/88/168 中为保留位，读返回值为 "0”。

- **位 4 – RWWSRE: RWW 区读使能**

RWW区处于编程(页擦除或页写入)状态时，RWW区的读操作(RWWSB被硬件置"1”)将被阻塞。用户软件必须等到编程结束(SPMEN清零)才能重新使能RWW区。如果 RWWSRE 位和SPMEN同时被写入"1”，则在紧接着的四个时钟周期内的SPM指令将再次使能RWW区。如果 Flash 忙于页擦除或页写入 (SPMEN 置位),RWW 区不能被使能。如果 Flash 加载与 RWWSRE 写操作同时发生，则 Flash 加载操作终止，加载的数据亦将丢失。

- **位 3 – BLBSET: Boot 锁定位设置**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令会根据 R0 中的数据设置 Boot 锁定位。R1 中的数据和 Z 指针的地址信息被忽略。锁定位设置完成，或在四个时钟周期内没有 SPM 指令被执行时， BLBSET 自动清零。

在 SPMCSR 寄存器的 BLBSET 和 SPMEN 置位后的三个周期内运行的 LPM 指令将读取锁定位或熔丝位 (取决于 Z 指针的 Z0) 并送到目的寄存器。详见 P249” 通过软件读取熔丝位和锁定位 ”。

- **位 2 – PGWRT: 页写入**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页写功能，将临时缓冲器中存储的数据写入 Flash。页地址取自 Z 指针的高位部分。R1 和 R0 的数据则被忽略。页写操作完成，或在四个时钟周期内没有 SPM 指令被执行时，PGWRT 自动清零。如果页写对象为 NRWW 区，在整个页写操作过程中 CPU 停止。

- **位 1 – PGERS: 页擦除位**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页擦除功能。页地址取自 Z 指针的高位部分。 R1 和 R0 的数据则被忽略。页擦除操作完成，或在四个时钟周期内没有SPM指令被执行时，PGERS自动清零。如果页写对象为NRWW区，在整个页擦除操作过程中 CPU 停止。

- **位 0 – SPMEN: 存贮程序存储器使能位**

这一位使能紧接着的四个时钟周期内的 SPM 指令。如果将这一位和 RWWSRE、BLBSET、PGWRT 或 PGERS 之一同时置位，则如上所述，接下来的 SPM 指令将有特殊的含义。如果只有 SPMEN 置位，那么接下来的 SPM 指令将把 R1:R0 中的数据存储到由 Z 指针确定的临时页缓冲器。Z 指针的 LSB 被忽略。SPM 指令完成，或在四个时钟周期内没有 SPM 指令被执行时， SPMEN 自动清零。在页擦除和页写过程中 SPMEN 保持为 1 直到操作完成。

在低五位中写入除 “10001”、“01001”、“00101”、 “00011” 或 “00001” 之外的任何组合都无效。

自编程过程中寻址 Flash

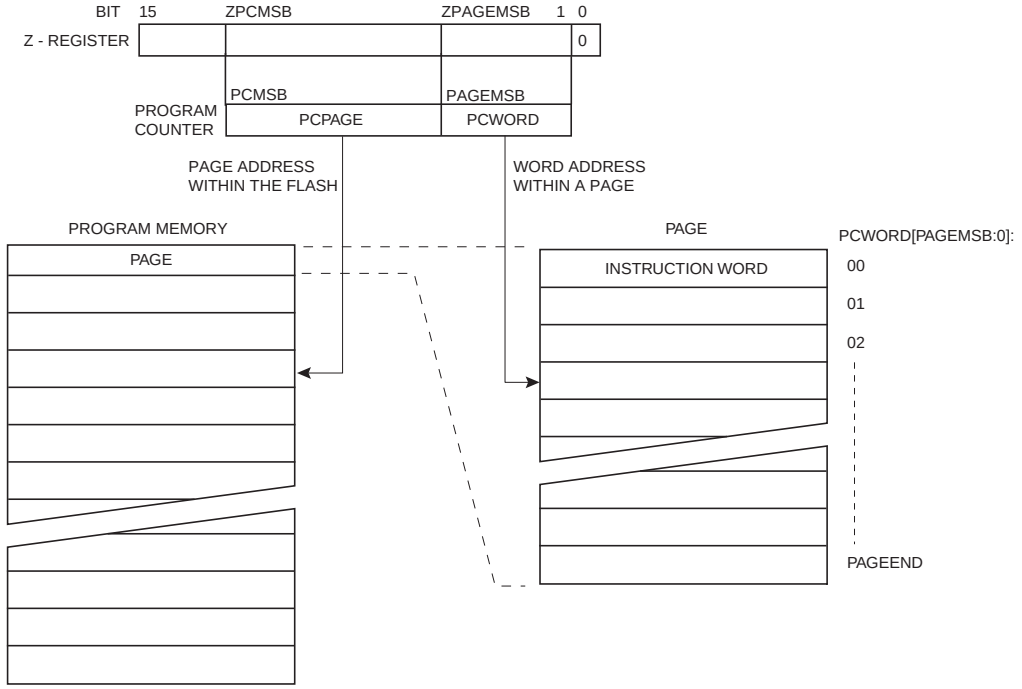
Z 指针用于 SPM 命令的寻址。

位	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

由于 Flash 存储器是以页的形式组织 (P261Table126) 起来的，程序计数器可看作由两个部分构成：其一为实现页内寻址的低位部分；其次为实现页寻址的高位部分，如 Figure118所示。由于页擦除和页写操作的寻址是相互独立的，因此保证Boot Loader软件在页擦除和页写操作时寻址相同的页是最重要的。一旦编程操作开始启动，地址就被锁存，然后 Z 指针可以用作其他用途了。

唯一不使用 Z 指针的 SPM 操作是设置 Boot Loader 锁定位。Z 指针的内容被忽略。LPM 指令也使用 Z 指针来保存地址。由于这个指令的寻址逐字节地进行，所以 Z 指针的 LSB 位 (位 Z0) 也使用到了。

Figure 118. SPM(1) 的寻址



Note: 1. Figure118 中所用的不同的变量在 P252Table111 列出。

自编程 Flash

程序存储器的更新以页的方式进行。在用临时页缓冲器存储的数据对一页存储器进行编程时，首先要将这一页擦除。 SPM 指令以一次一个字的方式将数据写入临时页缓冲器。临时页缓冲器的写入可以在页擦除命令之前完成，也可以在页擦除和页写操作之间完成。

方案 1，在页擦除前写缓冲器

- 写临时页缓冲器
- 执行页擦除操作
- 执行页写操作

方案 2，在页擦除后写缓冲器

- 执行页擦除操作
- 写临时页缓冲器

- 执行页写操作

如果只需要改变页的一部分，则在页擦除之前必须将页中其他部分存储起来（如保存于临时页缓冲区中），然后再写回 Flash。使用方案 1 时，Boot Loader 提供了一个有效的读 - 修改 - 写特性，允许用户软件首先读取页中的内容，然后对内容做必要的改变，接着把修改后的数据写回 Flash。如果使用方案 2，则无法读取旧数据，因为页已经被擦除了。临时页缓冲区可以随机寻址。保证在页擦除和页写操作中寻址相同的页是很关键的。汇编代码的例子请参见 P251” 简单的 Boot Loader 汇编代码例子 ”。

利用 SPM 进行页擦除

执行页擦除操作首先需要设置 Z 指针的地址信息，然后将 “X0000011” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 擦除 RWW 区的页：在页擦除过程中可以读取 NRWW 区
- 擦除 NRWW 区的页：在操作过程中 CPU 停止

写临时缓冲区 (页加载)

写一个指令字首先需要设置 Z 指针的地址信息，以及将指令字写入 R1:R0，然后将 “00000001” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。Z 寄存器中 PCWORD 的内容用来寻址临时缓冲区。页写操作完成，或置位 SPMCSR 寄存器的 RWWSRE 将使临时缓冲区自动擦除。系统复位也会擦除临时缓冲区。但是如果不清除临时缓冲区就只能对每个地址进行一次写操作。

如果在 SPM 页加载操作过程中对 EEPROM 执行了写操作，则所有加载的数据都将丢失。

执行页写

执行页写操作首先需要设置 Z 指针的地址信息，然后将 “X0000101” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 擦除 RWW 区的页：在页擦除过程中可以读取 NRWW 区
- 擦除 NRWW 区的页：在页写过程中 CPU 停止

使用 SPM 中断

如果 SPM 中断使能，则 SPMCSR 寄存器的 SPMEN 清零将产生中断。这意味着软件可以利用中断来代替对 SPMCSR 寄存器的查询。使用 SPM 中断时，要将中断向量移到 BLS，以避免 RWW 区读禁止时中断程序却访问它。如何移动中断向量请见 P44” 看门狗定时器 ”。

更新 BLS 时要考虑的问题

通过不编程 Boot 锁定位 11 的方式来更新 Boot Loader 区时需要给予格外关注。对 Boot Loader 本身进行的误操作会破坏整个 Boot Loader ，造成软件无法更新。如果程序不需要改变 Boot Loader，建议对 Boot 锁定位 11 编程，以防止不小心改变了 Boot Loader。

防止自编程过程中访问 RWW 区

在自编程过程中 (页擦除或页写) , 对 RWW 区的访问被阻塞。用户软件要避免此情况发生。RWW 区忙将使 SPMCSR 寄存器的 RWWSB 置位。在自编程时，如 P44” 看门狗定时器 ” 所述，中断向量表应该移到 BLS 中，或者禁止中断。编程结束后，在寻址 RWW 区之前用户软件必须对 RWWSRE 写 1 来清零 RWWSB。例子请见 P251” 简单的 Boot Loader 汇编代码例子 ”。

通过 SPM 设置 Boot Loader 锁定位

设置Boot Loader 锁定位首先要给R0赋予期望的数据，然后将“X0001001”写入SPMCSR寄存器，并在紧接着的四个时钟周期内执行 SPM 指令。唯一可访问的锁定位是 Boot Loader 锁定位。利用这个锁定位可以阻止 MCU 对应用程序和 Boot Loader 软件的更新。

位	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

不同的 Boot Loader 锁定位设置对 Flash 访问的影响请参见 Table105 和 Table106。

如果 R0 的 5..2 位为 0，并且在 SPMCSR 寄存器的 BLBSET 和 SPMEN 置位之后的四个周期内执行了 SPM 指令，相应的 Boot 锁定位将被编程。此操作不使用 Z 指针，但出于兼容性的考虑，建议将 Z 指针赋值为 0x0001(与读 IO_{ck} 位的操作相同)。同样出于兼容性的考虑，建议在写锁定位时将 R0 中的 7、6、 1 和第 0 位置 "1”。在编程锁定位的过程中可以自由访问整个 Flash 区。

EEPROM 写操作对 SPMCSR 寄存器写操作的阻塞

EEPROM写操作会阻塞对Flash的编程，也会阻塞对熔丝位和锁定位的读操作。建议用户在对 SPMCSR 寄存器进行写操作之前首先检查 EECR 寄存器的状态位 EEWE，确保此位以被清除。

通过软件读取熔丝位和锁定位

熔丝位和锁定位可以通过软件读取。读锁定位时，需要将 0x0001 赋予给 Z 指针并且置位SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的 LPM 指令将把锁定位的值将加载到目的寄存器。读锁定位操作结束，或者在三个 CPU 周期内没有 执行LPM指令，或在四个CPU周期内没有执行SPM指令，BLBSET 和 SPMEN 位将自动硬件清零。 BLBSET 和 SPMEN 清零后，LPM 将按照指令手册中所描述的那样工作。

位	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

读取熔丝位低字节的算法和上述读取锁定位的算法类似。要读取熔丝位低字节，需要将0x0000赋予给Z指针并且置位SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的LPM指令将把熔丝位低位字节的值 (FLB) 加载到目的寄存器。更详细的说明及熔丝位低位字节映射的细节请参见 P256Table119 。

位	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

类似的，读取熔丝位高位字节时，需要将 0x0003 赋予给 Z 指针并且置位 SPMCSR 寄存器的 BLBSET 和 SPMEN。在 SPMCSR 操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位高位字节的值 (FHB) 加载到目的寄存器。更详细的说明及熔丝位高位字节映射的细节请参见 P256Table120 。

位	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

读取熔丝位扩展字节时，需要将 0x0002 赋予给 Z 指针并且置位 SPMCSR 寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的LPM指令将把熔丝位扩展字节的值 (EFB) 加载到目的寄存器。更详细的说明及熔丝位扩展字节映射的细节请参见 P256Table118 。

位	7	6	5	4	3	2	1	0
Rd	–	–	–	–	EFB3	EFB2	EFB1	EFB0

被编程的熔丝位和锁定位的读返回值为 "0”。未被编程的熔丝位和锁定位的读返回值为 "1”。

防止 Flash 损毁

V_{CC} 低于工作电压时，CPU 和 Flash 正常工作无法保证，Flash 的内容可能受到破坏。这个问题对于应用于板级系统的独立 Flash 一样存在。所以也要采用同样的解决方案。

电压太低时有两种情况可以破坏 Flash 内容。第一，Flash 写过程需要一个最低电压。第二，电压太低时 CPU 本身会错误地执行指令。

通过遵循以下设计建议可以避免 Flash 被破坏 (采用其中之一就足够了) :

1. 如果系统不需要更新 Boot Loader，建议编程 Boot Loader 锁定位以防止 Boot Loader 软件更新
2. 电源电压不足期间，保持 AVR RESET 为低：采用的方式为：如果工作电压与检测电平相匹配，可以使能 BOD 功能；否则可以使用外部复位保护电路。如果在写操作进行中发生了复位，只要电源电压足够，写操作还会完成。
3. 低电压期间保持 AVR 内核处于掉电休眠模式。这样可以防止 CPU 解码并执行指令，有效地保护 SPMCSR 寄存器，从而保护 Flash 被无意识得修改掉。

使用 SPM 时的 Flash 编程时间

片内校准的 RC 振荡器用于 Flash 寻址时序控制。Table108 给出了 CPU 访问 Flash 的典型编程时间。

Table 108. SPM 编程时间

符号	最小编程时间	最大编程时间
Flash 写操作 (通过 SPM 实现页擦除、页写、及写锁定位)	3.7 ms	4.5 ms

简单的 Boot Loader 汇编代码例子

```
  ;- 本例程将 RAM 中的一页数据写入 Flash
  ; Y 指针指向 RAM 的第一个数据单元
  ;Z 指针指向 Flash 的第一个数据单元
  ;- 本例程没有包括错误处理
  ;- 该程序必须放置于 Boot 区 ( 至少 Do_spm 子程序是如此 )
  ; 在自编程过程中 ( 页擦除和页写操作 ) 只能读访问 NRWW 区的代码
  ;- 使用的寄存器：r0、 r1、 temp1 (r16)、 temp2 (r17)、 looplo (r24)、
  ; loophi (r25)、 spmcrval (r20)
  ; 在程序中不包括寄存器内容的保护和恢复
  ; 在牺牲代码大小的情况下可以优化寄存器的使用
  ;- 假设中断向量表位于 Boot loader 区，或者中断被禁止。
.equ  PAGESIZEB = PAGESIZE*2      ;PAGESIZEB 是以字节为单位的页大小，不是以字为单位
.org SMALLBOOTSTART
Write_page:
  ; 页擦除
  ldi   spmcrval, (1<<PGERS) | (1<<SPMEN)
  call  Do_spm

  ; 重新使能 RWW 区
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; 将数据从 RAM 转移到 Flash 页缓冲区
  ldi   looplo, low(PAGESIZEB)      ; 初始化循环变量
  ldi   loophi, high(PAGESIZEB)     ;PAGESIZEB<=256 时不需要此操作
Wrloop:
  ld    r0, Y+
  ld    r1, Y+
  ldi   spmcrval, (1<<SPMEN)
  call  Do_spm
  adiw  ZH:ZL, 2
  sbiw  loophi:looplo, 2            ;PAGESIZEB<=256 时请使用 subi
  brne  Wrloop

  ; execute Page Write
  subi  ZL, low(PAGESIZEB)          ; 复位指针
  sbci  ZH, high(PAGESIZEB)         ;PAGESIZEB<=256 时不需要此操作
  ldi   spmcrval, (1<<PGWRT) | (1<<SPMEN)
  call  Do_spm

  ; 重新使能 RWW 区
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; 读回数据并检查，为可选操作
  ldi   looplo, low(PAGESIZEB)      ; 初始化循环变量
  ldi   loophi, high(PAGESIZEB)     ;PAGESIZEB<=256 时不需要此操作
  subi  YL, low(PAGESIZEB)          ; 复位指针
  sbci  YH, high(PAGESIZEB)
Rdloop:
  lpm   r0, Z+
  ld    r1, Y+
  cpse  r0, r1
  jmp   Error
  sbiw  loophi:looplo, 1            ;PAGESIZEB<=256 时请使用 subi
  brne  Rdloop

  ; 返回到 RWW 区
  ; 确保 RWW 区已经可以安全读取
Return:
  in    temp1, SPMCSR
  sbrs  temp1, RWWSB       ; 若 RWWSB 为 "1"，说明 RWW 区还没有准备好
  ret
  ; 重新使能 RWW 区
```

```
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm
  rjmp  Return

Do_spm:
  ; 检查先前的 SPM 操作是否已经完成
Wait_spm:
  in    temp1, SPMCSR
  sbrc  temp1, SPMEN
  rjmp  Wait_spm
  ; 输入：spmcrval 决定了 SPM 操作
  ; 禁止中断，保存状态标志
  in    temp2, SREG
  cli
  ; 确保没有 EEPROM 写操作
Wait_ee:
  sbic  EECR, EEPE
  rjmp  Wait_ee
  ; SPM 时间序列
  out   SPMCSR, spmcrval
  spm
  ; 恢复 SREG ( 如果中断原本是使能的，则使能中断 )
  out   SREG, temp2
  ret
```

ATmega88 Boot Loader 参数

自编程描述中所用的参数在 Table109 到 Table111 中给出。

Table 109. ATmega88 Boot 区大小配置 (1)

BOOTSZ1	BOOTSZ0	Boot 区大小	页数	应用 Flash 区	Boot Loader Flash 区	应用区结束地址	Boot 复位地址 (Boot Loader 起始地址)
1	1	128 字	4	0x000 - 0xF7F	0xF80 - 0xFFF	0xF7F	0xF80
1	0	256 字	8	0x000 - 0xEFF	0xF00 - 0xFFF	0xEFF	0xF00
0	1	512 字	16	0x000 - 0xDFF	0xE00 - 0xFFF	0xDFF	0xE00
0	0	1024 字	32	0x000 - 0xBFF	0xC00 - 0xFFF	0xBFF	0xC00

Note: 不同的 BOOTSZ 熔丝位配置请参见 Figure117.

Table 110. ATmega88 RWW 界限 (1)

Flash 区	页数	寻址范围
同时读 - 写区 (RWW)	96	0x000 - 0xBFF
非同时读 - 写区 (NRWW)	32	0xC00 - 0xFFF

关于两个区的详细说明请见 P242”NRWW – 非同时读 - 写存储区 ” 及 P241”RWW – 同时读 - 写存储区 ” 。

Table 111. ATmega88 Figure118 中所用变量的说明及 Z 指针的映射 (1)

变量		相应的 Z 指针数据	描述
PCMSB	11		程序计数器的最高位 (程序计数器为 12 位 PC[11:0])
PAGEMSB	4		用于页内字寻址的最高位 (一页有 32 个字，需要 5 位 PC [4:0]).
ZPCMSB		Z12	Z 寄存器与 PCMSB 对应的位。由于没有使用 Z0 ， ZPCMSB 等于 PCMSB + 1
ZPAGEMSB		Z5	Z 寄存器与 PAGEMSB 对应的位。由于没有使用 Z0 ， ZPAGEMSB 等于 PAGEMSB + 1
PCPAGE	PC[11:5]	Z12:Z6	程序计数器页地址：在页擦除和页写操作中进行页选择
PCWORD	PC[4:0]	Z5:Z1	程序计数器字地址：为填充临时缓冲区进行字选择 (在页写过程中必须为 0)

Note: 1. Z15:Z13：始终忽略
Z0：对所有的 SPM 命令都为 "0”，对 LPM 指令则用于字节选择。
关于自编程过程中 Z 指针的使用请参见 P246” 自编程过程中寻址 Flash” 。

ATmega168 Boot Loader 参数

自编程描述中所用的参数在 Table112 到 Table114 中给出。

Table 112. ATmega168 Boot 区大小配置 (1)

BOOTSZ1	BOOTSZ0	Boot 区大小	页数	应用 Flash 区	Boot Loader Flash 区	应用区结束地址	Boot 复位地址 (Boot Loader 起始地址)
1	1	128 字	4	0x0000 - 0x1F7F	0x1F80 - 0x1FFF	0x1F7F	0x1F80
1	0	256 字	8	0x0000 - 0x1EFF	0x1F00 - 0x1FFF	0x1EFF	0x1F00
0	1	512 字	16	0x0000 - 0x1DFF	0x1E00 - 0x1FFF	0x1DFF	0x1E00
0	0	1024 字	32	0x0000 - 0x1BFF	0x1C00 - 0x1FFF	0x1BFF	0x1C00

Note: 不同的 BOOTSZ 熔丝位配置请参见 Figure117。

Table 113. ATmega168 RWW 界限 (1)

Flash 区	页数	寻址范围
同时读 - 写区 (RWW)	112	0x0000 - 0x1BFF
非同时读 - 写区 (NRWW)	16	0x1C00 - 0x1FFF

关于两个区的详细说明请见 P242”NRWW – 非同时读 - 写存储区 ” 及 P241”RWW – 同时读 - 写存储区 ”。

Table 114. ATmega168 Figure118 中所用变量的说明及 Z 指针的映射 (1)

变量		相应的 Z 指针数据	描述
PCMSB	12		程序计数器的最高位 (程序计数器为 12 位 PC[11:0])
PAGEMSB	5		用于页内字寻址的最高位 (一页有 32 个字，需要 5 位 PC [4:0]).
ZPCMSB		Z13	Z 寄存器与 PCMSB 对应的位。由于没有使用 Z0 ， ZPCMSB 等于 PCMSB + 1
ZPAGEMSB		Z6	Z 寄存器与 PAGEMSB 对应的位。由于没有使用 Z0 ， ZPAGEMSB 等于 PAGEMSB + 1
PCPAGE	PC[12:6]	Z13:Z7	程序计数器页地址：在页擦除和页写操作中进行页选择
PCWORD	PC[5:0]	Z6:Z1	程序计数器字地址：为填充临时缓冲区进行字选择 (在页写过程中必须为 0)

Note: 1. Z15:Z14：始终忽略
Z0：对所有的 SPM 命令都为 "0”，对 LPM 指令则用于字节选择。
关于自编程过程中 Z 指针的使用请参见 P246” 自编程过程中寻址 Flash” 。

存储器编程

程序存储器和数据存储器锁定位

ATmega88/168 提供了6个锁定位，根据其被编程(“0”)还是没有被编程 (“1”)的情况可以获得 Table116 列出的附加性能。锁定位只能通过芯片擦除命令擦写为 “1”。ATmega48 没有独立的 Boot Loader 区。如果 SPMEN 熔丝位被编程 (“0”)，SPM 指令可以在整个 Flash 区中运行；否则 SPM 指令被禁用。

Table 115. 锁定位字节 (1)

锁定位字节	位号	描述	默认值
	7	–	1 (未编程)
	6	–	1 (未编程)
BLB12	5	Boot 锁定位	1 (未编程)
BLB11	4	Boot 锁定位	1 (未编程)
BLB02	3	Boot 锁定位	1 (未编程)
BLB01	2	Boot 锁定位	1 (未编程)
LB2	1	锁定位	1 (未编程)
LB1	0	锁定位	1 (未编程)

Note: 1. “1” 表示未编程，“0” 表示被编程。

Table 116. 锁定位保护模式 (1)(1)

存储器锁定位			保护类型
LB 模式	LB2	LB1	
1	1	1	没有使能存储器保护特性
2	1	0	在并行和串行编程模式中 Flash 和 EEPROM 的进一步编程被禁止，熔丝位被锁定。(1)
3	0	0	在并行和串行编程模式中 Flash 和 EEPROM 的进一步编程及验证被禁止，锁定位和熔丝位被锁定 (1)

Notes: 1. 在编程 LB1 和 LB2 前先编程熔丝位和 Boot 锁定位。
Notes: 1. “1” 表示未被编程，“0” 表示被编程。

Table 117. ATmega88/168 锁定位保护模式 (1)(1)

BLB0 模式	BLB02	BLB01	
1	1	1	SPM 和 LPM 对应用区的访问没有限制
2	1	0	不允许 SPM 对应用区进行写操作
3	0	0	不允许 SPM 指令对应用区进行写操作，也不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
BLB1 模式	**BLB12**	**BLB11**	
1	1	1	允许 SPM/LPM 指令访问 Boot Loader 区
2	1	0	不允许 SPM 指令对 Boot Loader 区进行写操作
3	0	0	不允许 SPM 指令对 Boot Loader 区进行写操作，也不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。

Notes: 1. 在编程 LB1 和 LB2 前先编程熔丝位和 Boot 锁定位。
2. “1” 表示未被编程 , “0” 表示被编程。

熔丝位

ATmega48/88/168 有三个熔丝位字节。Table118 - Table120 简单地描述了所有熔丝位的功能以及他们是如何映射到熔丝字节的。如果熔丝位被编程则读返回值为 “0”。

Table 118. mega48 熔丝位扩展字节

熔丝位低字节	位号	描述	默认值
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
–	1	–	1
SPMEN	0	SPM 使能	1 (未被编程)

Notes: 1. BODLEVEL 熔丝位对应的具体电平请参见 P39Table21 。
2. RSTDISBL 熔丝位的描述见 P69” 端口 C 的第二功能 ” 。

Table 119. mega88/168 熔丝位扩展字节

熔丝位低字节	位号	描述	默认值
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
BOOTSZ1	2	选择 Boot 区大小 (详见 Table 113)	0 (已编程)(1)
BOOTSZ0	1	选择 Boot 区大小 (详见 Table 113)	0 (已编程)(1)
BOOTRST	0	选择复位向量	1 (未被编程)

Note: 1. BOOTSZ1..0 默认值给出 Boot 最大值，详见 P259Table122 。

Table 120. 熔丝位高位字节

熔丝位高位字节	位号	描述	默认值
RSTDISBL(1)	7	外部复位禁用	1 (未被编程)
DWEN	6	调试线使能	1 (未被编程)
SPIEN(2)	5	使能串行程序和数据下载	0 (被编程，SPI 编程使能)
WDTON(3)	4	看门狗定时器一直启用	1 (未被编程)
EESAVE	3	执行芯片擦除时 EEPROM 的内容保留	1 (未被编程) , EEPROM 内容不保留
BODLEVEL2(4)	2	BOD 触发电平	1 (未被编程)
BODLEVEL1(4)	1	BOD 触发电平	1 (未被编程)
BODLEVEL0(4)	0	BOD 触发电平	1 (未被编程)

Notes: 1. RSTDISBL 熔丝位的说明，请见 P69” 端口 C 的第二功能 ” 。

2. 在串行编程模式下 SPIEN 熔丝位不可访问。
3. 详细内容见 P44” 看门狗定时器控制寄存器 – WDTCSR” 。
4. P39Table21 给出 BODLEVEL 熔丝位具体设置。

Table 121. 熔丝位低位字节

熔丝位低位字节	位号	描述	默认值
CKDIV8(4)	7	时钟 8 分频	0 (被编程)
CKOUT(3)	6	时钟输出	1 (未被编程)
SUT1	5	选择启动时间	1 (未被编程)(1)
SUT0	4	选择启动时间	0 (被编程)(1)
CKSEL3	3	选择时钟源	0 (被编程)(2)
CKSEL2	2	选择时钟源	0 (被编程)(2)
CKSEL1	1	选择时钟源	1 (未被编程)(2)
CKSEL0	0	选择时钟源	0 (被编程)(2)

Note: 1. 对于默认时钟源 ,SUT1..0 的默认值给出最大的起动时间。详细内容见 P28Table12 。
2. CKSEL3..0的默认设置导致了片内RC振荡器运行于8 MHz。详细内容见 P27Table11 。
3. CKOUT 熔丝位允许系统时钟从 PORTB0 输出。详细内容见 P31” 时钟输出缓冲器 ” 。
4. 详细内容见 P31” 系统时钟预分频器 ” 。

熔丝位的状态不受芯片擦除命令的影响。如果锁定位 1(LB1) 被编程则熔丝位被锁定。在编程锁定位前先编程熔丝位。

熔丝位的锁存

器件进入编程模式时熔丝位的值被锁存。其间熔丝位的改变不会生效 , 直到器件退出编程模式。不过这不适用于 EESAVE 熔丝位。它一旦被编程立即起作用。在正常工作模式中器件上电时熔丝位也被锁存。

标识字节

所有的 Atmel 微控制器都具有一个三字节的标识代码用来区分器件型号。这个代码可以通过串行和并行模式读取，也可以在芯片被锁定时读取。这三个字节分别存储于三个独立的地址空间。

ATmega48 标识字节

1. 0x000: 0x1E (表示由 Atmel 公司生产)
2. 0x001: 0x92 (表示芯片包含 4KB Flash 存储器)
3. 0x002: 0x05 (当 0x001 字节的内容为 0x92 时表示这是 ATmega48)

ATmega88 标识字节

1. 0x000: 0x1E (表示由 Atmel 公司生产)
2. 0x001: 0x93 (表示芯片包含 8KB Flash 存储器)
3. 0x002: 0x0A (当 0x001 字节的内容为 0x93 时表示这是 ATmega88)

ATmega168 标识字节

1. 0x000: 0x1E (表示由 Atmel 公司生产)
2. 0x001: 0x94 (表示芯片包含 16KB Flash 存储器)
3. 0x002: 0x06 (当 0x001 字节的内容为 0x94 时表示这是 ATmega168)

校准字节

ATmega48/88/168 内部 RC 振荡器的校准值保存于校准字节。这个字节位于标识地址空间 0x000 的高位字节。在复位期间，该字节被自动写入 OSCCAL 寄存器以确保校准的 RC 振荡器频率的正确性。

并行编程参数，引脚映射及命令

这部分描述了如何对 ATmega48/88/168 的 Flash 程序存储器、 EEPROM 数据存储器、存储锁定位及熔丝位进行并行编程和校验。除非另有说明，否则脉冲宽度至少为 250 ns。

信号名称

在这一节 ATmega48/88/168 的相关引脚以并行编程信号的名称进行引用，如 Figure119 和 Table122 所示。表中没有描述的引脚沿用原来的称谓。

XA1/XA0 决定了给 XTAL1 引脚一个正脉冲时所执行的操作。具体编码请见 Table124。

给 $\overline{WR}$ 或 $\overline{OE}$ 输入脉冲时所加载的命令决定了要执行的操作。具体命令请参见 Table125。

Figure 119. 并行编程

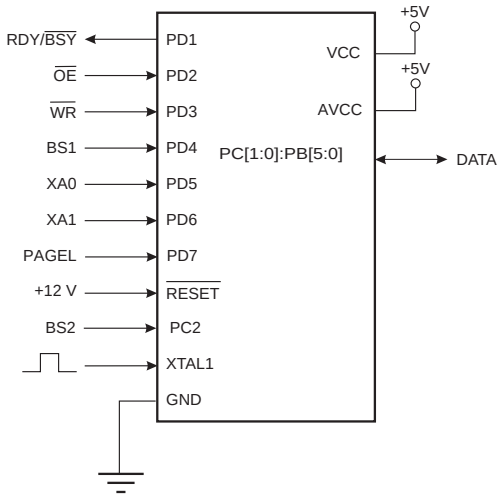


Table 122. 引脚名称映射

编程模式信号的名称	引脚名称	I/O	功能
RDY/$\overline{BSY}$	PD1	O	0: 设备忙于编程，1: 设备等待新的命令。
$\overline{OE}$	PD2	I	输出使能 (低电平有效).
$\overline{WR}$	PD3	I	写脉冲 (低电平有效).
BS1	PD4	I	字节选择 1(“0” 选择低位字节，“1” 选择高位字节).
XA0	PD5	I	XTAL 动作位 0
XA1	PD6	I	XTAL 动作位 1
PAGEL	PD7	I	加载程序存储器和 EEPROM 数据页
BS2	PC2	I	字节选择 2(“0” 选择低位字节，“1” 选择第二个高位字节)
DATA	{PC[1:0]: PB[5:0]}	I/O	双向数据总线 ($\overline{OE}$ 为低时输出)

Table 123. 进入编程模式所需要的引脚数据

引脚	符号	数值
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Table 124. XA1 和 XA0 的编码

XA1	XA0	给 XTAL1 施加脉冲激发的操作
0	0	加载 Flash 或 EEPROM 地址 (通过 BS1 确定是高位还是低位字节)
0	1	加载数据 (通过 BS1 决定是高位还是低位闪存数据字节)
1	0	加载命令
1	1	无操作，空闲

Table 125. 命令字节编码

命令字节	执行的命令
1000 0000	芯片擦除
0100 0000	写熔丝位
0010 0000	写锁定位
0001 0000	写 Flash
0001 0001	写 EEPROM
0000 1000	读标识字节和校准字节
0000 0100	读熔丝位和锁定位
0000 0010	读 Flash
0000 0011	读 EEPROM

Table 126. 一页包含的字和 Flash 中的页数

器件	Flash 大小	页大小	PCWORD	页号	PCPAGE	PCMSB
ATmega48	2K 字 (4K 字节)	32 字	PC[4:0]	64	PC[10:5]	10
ATmega88	4K 字 (8K 字节)	32 字	PC[4:0]	128	PC[11:5]	11
ATmega168	8K 字 (16K 字节)	64 字	PC[5:0]	128	PC[12:6]	12

Table 127. 一页包含的字和 EEPROM 中的页数

器件	EEPROM 大小	页大小	PCWORD	页数	PCPAGE	EEAMSB
ATmega48	256 字节	4 字节	EEA[1:0]	64	EEA[7:2]	7
ATmega88	512 字节	4 字节	EEA[1:0]	128	EEA[8:2]	8
ATmega168	512 字节	4 字节	EEA[1:0]	128	EEA[8:2]	8

串行编程引脚映射

Table 128. 串行编程时的引脚映射

符号	引脚	I/O	描述
MOSI	PB3	I	串行数据输入
MISO	PB4	O	串行数据输出 t
SCK	PB5	I	串行时钟

并行编程

进入编程模式

通过下面的算法进入并行编程模式：

1. 在 V_{CC} 及 GND 之间提供 4.5 - 5.5V 的电压
2. 将 $\overline{\text{RESET}}$ 拉低，并至少改变 XTAL1 电平 6 次
3. 将 P260Table123 中列出的的 Prog_enable 引脚置为 "0000"，并等待至少 100 ns
4. 给 $\overline{\text{RESET}}$ 提供 11.5 - 12.5V 的电压。在向 $\overline{\text{RESET}}$ 提供 +12V 电压后的 100 ns 内，Prog_enable 引脚的任何行为都会导致芯片无法进入编程模式
5. 在发送新命令之前至少等待 50 µs

高效编程的几点考虑

在编程过程中，加载的命令及地址保持不变。为了实现高效的编程应考虑以下因素：

- 对多个存储单元进行读或写操作时，命令仅需加载一次
- 当需要写入的数据为 0xFF 时可以跳过，因为这就是执行全片擦除命令后 Flash 及 EEPROM(除非 EESAVE 熔丝位被编程) 的内容。
- 只有在编程或读取Flash及EEPROM中新的256字时才需要用到地址高位字节。在读标识字节时也需考虑这一点。

芯片擦除

芯片擦除操作会擦除 Flash 及 EEPROM(1) 存储器以及锁定位。程序存储器没有擦除结束之前锁定位不会复位。全片擦除不影响熔丝位。芯片擦除命令必须在编程 Flash 与 / 或 EEPROM 之前完成。

Note: 1. 如果 EESAVE 熔丝位被编程，那么在芯片擦除时 EEPRPOM 不受影响。

加载 " 芯片擦除 " 命令的过程：

1. 将 XA1、 XA0 置为 10 以启动命令加载
2. 将 BS1 置为 0
3. DATA 赋值为 “1000 0000”。这是芯片擦除命令
4. 给 XTAL1 提供一个正脉冲，进行命令加载
5. 给 $\overline{\text{WR}}$ 提供一个负脉冲，启动芯片擦除。 RDY/$\overline{\text{BSY}}$ 变低
6. 等待 RDY/$\overline{\text{BSY}}$ 变高，然后才能加载新的命令

对 Flash 进行编程

Flash 是以页的形式组织起来的，如 P261Table126 所示。编程 Flash 时，程序数据被锁存到页缓冲区中。这样一整页的程序数据可以同时得到编程。下面的步骤描述了如何对 Flash 进行编程：

A. 加载 " 写 Flash" 命令

1. 将 XA1、 XA0 置为 "10"，启动命令加载
2. 将 BS1 置 "0”
3. DATA 赋值为 “0001 0000”，这是写 Flash 命令
4. 给 XTAL1 提供一个正脉冲以加载命令

B. 加载地址低位字节

1. 将 XA1、 XA0 置为 "00"，启动地址加载
2. 将 BS1 置 "0”，选择低位地址
3. DATA 赋值为地址低位字节 (0x00 - 0xFF)
4. 给 XTAL1 提供一个正脉冲，加载地址低位字节

C. 加载数据低位字节

1. 将 XA1、 XA0 置为 "01"，启动数据加载
2. DATA 赋值为数据低位字节 (0x00 - 0xFF)

3. 给 XTAL1 提供一个正脉冲，加载数据字节

D. 加载数据高位字节

1. 将 BS1 置为 "1”，选择数据高位字节
2. 将 XA1、 XA0 置为 "01"，启动数据加载
3. DATA 赋值为数据高位字节 (0x00 - 0xFF)
4. 给 XTAL1 提供一个正脉冲，进行数据字节加载

E. 锁存数据

1. 将 BS1 置为 "1”，选择数据高位字节
2. 给 PAGEL 提供一个正脉冲，锁存数据 (见 Figure121 信号波形)

F. 重复 B 到 E 操作，直到整个缓冲区填满或此页中所有的数据都已加载

地址信息中的低位用于页内寻址，高位用于 FLASH 页的寻址，详见 P264Figure 120 。如果页内寻址少于 8 位 (页地址 < 256)，那么进行页写操作时地址低字节中的高位用于页寻址。

G. 加载地址高位字节

1. 将 XA1、 XA0 置为 "00”，启动地址加载操作
2. 将 BS1 置为 "1”，选择高位地址
3. DATA 赋值为地址高位字节 (0x00 - 0xFF)
4. 给 XTAL1 提供一个正脉冲，加载地址高位字节

H. 编程一页数据

1. 给 $\overline{WR}$ 提供一个负脉冲，对整页数据进行编程， RDY/$\overline{BSY}$ 变低
2. 等待 RDY/$\overline{BSY}$ 变高 (见 Figure121 的信号波形)

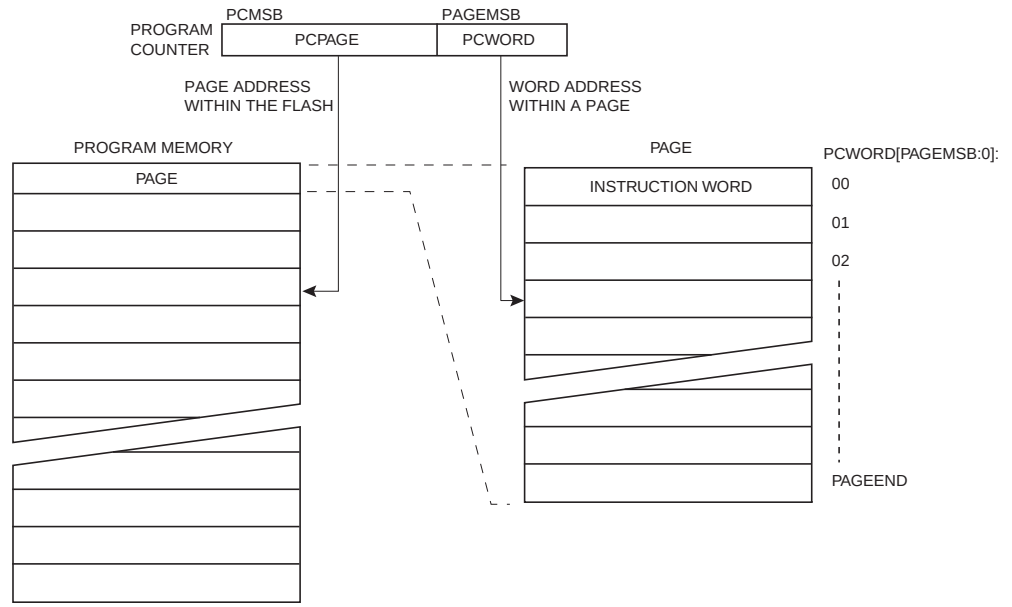
I. 重复 B 到 H 的操作，直到整个 Flash 编程结束或者所有的数据都被编程

J. 结束页编程

1. 1. 将 XA1、 XA0 置为 "10”，启动命令加载操作
2. DATA 赋值为 "0000 0000"，这是不操作指令

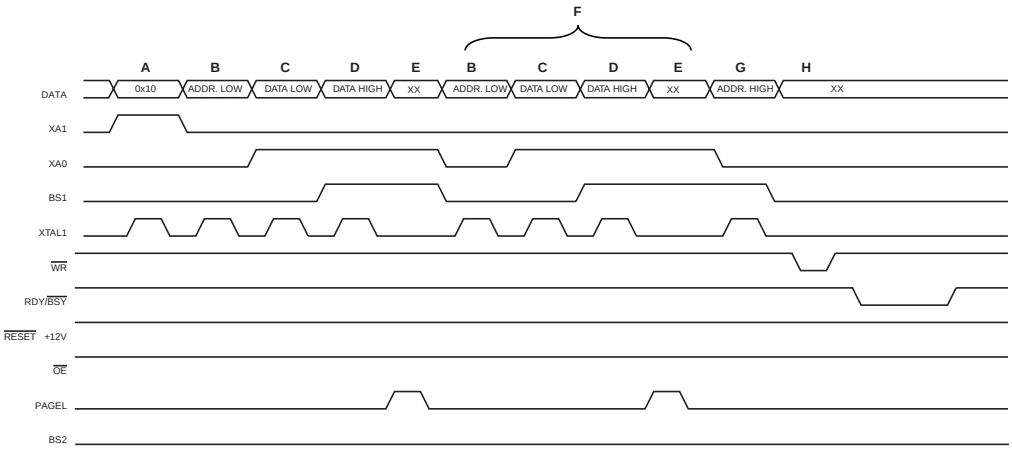
给 XTAL1 提供一个正脉冲，加载命令，内部写信号复位

Figure 120. 对以页为组织单位的 Flash 进行寻址 (1)



Note: 1. PCPAGE 及 PCWORD 列于 P261Table126 。

Figure 121. Flash 编程波形 (1)



Note: 1. 不用考虑 "XX"，各个大写字母对应于前面描述的 Flash 编程阶段。

对 EEPROM 进行编程

如 P261Table127 所示， EEPROM 也以页为单位。编程 EEPROM 时，编程数据锁存于页缓冲区中。这样可以同时对一页数据进行编程。 EEPROM 数据存储器编程算法如下 (命令、地址及数据加载的细节请参见 P263” 对 Flash 进行编程 ”) :

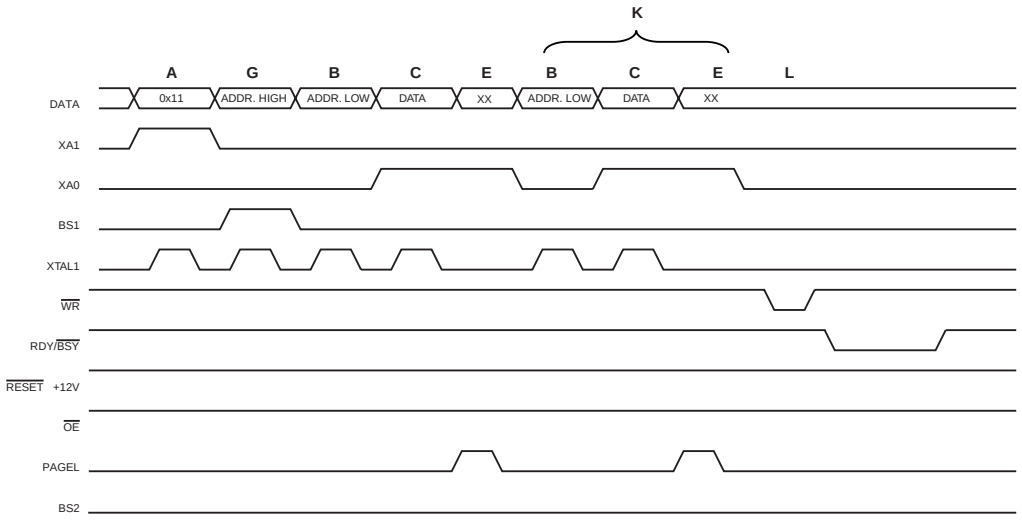
1. A：加载命令 “0001 0001”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. C：加载数据 (0x00 - 0xFF)
5. E：锁存数据 (给 PAGEL 提供一个正脉冲)

K：重复步骤 3 到 5，直到整个缓冲区填满

L：对 EEPROM 页进行编程

1. 将 BS 置 0
2. 给 $\overline{WR}$ 提供一个负脉冲，开始对 EEPROM 页进行编程， RDY/$\overline{BSY}$ 变低
3. 等到 RDY/$\overline{BSY}$ 变高再对下一页进行编程 (信号波形见 Figure122)

Figure 122. EEPROM 编程波形



读取 Flash 的内容

读 Flash 存储器的过程如下 (命令及地址加载细节见 P263” 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 0010”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. 将 $\overline{OE}$ 置 “0”， BS1 置 “0”，然后从 DATA 读出 Flash 字的低位字节
5. 将 BS 置 “1”，然后从 DATA 读出 Flash 字的高位字节
6. 将 $\overline{OE}$ 置 “1”。

读取 EEPROM 的内容

读存储器的步骤如下 (命令及地址加载细节见 P263” 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 0011”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. 将 $\overline{OE}$ 置 "0”， BS1 置 "0”，然后从 DATA 读出 EEPROM 数据字节
5. 将 $\overline{OE}$ 置 ”1”

对熔丝低位进行编程

对熔丝低位的编程步骤如下 (命令及数据装在细节见 P263” 对 Flash 进行编程 ”) :

1. A : 加载命令 “0100 0000”
2. C : 加载数据低字节 , 若某一位为 "0” 表示需要进行编程 , 否则需要擦除
3. 给 $\overline{WR}$ 提供一个负脉冲 , 并等待 RDY/$\overline{BSY}$ 变高

对熔丝高位进行编程

对熔丝高位的编程步骤如下 (命令及数据装在细节见 P263” 对 Flash 进行编程 ”) :

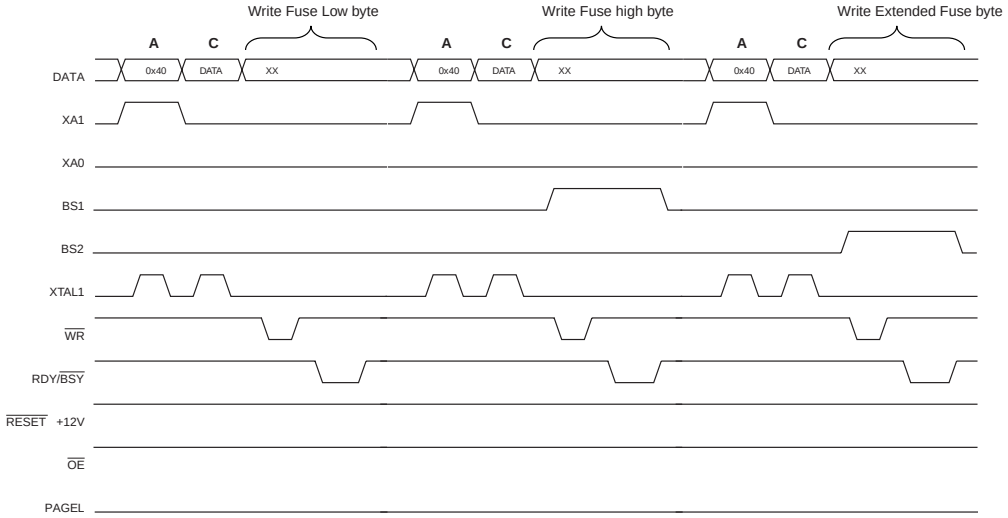
1. A : 加载命令 “0100 0000”
2. C : 加载数据高字节 , 若某一位为 "0” 表示需要进行编程 , 否则需要擦除
3. 将 BS1 置 "1”、 BS2 置 "0” , 选择高位数据字节
4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高
5. 将 BS1 置 "0” , 选择低位字节

对扩展熔丝位进行编程

对扩展熔丝位的编程步骤如下 (命令及数据装在细节见 P263” 对 Flash 进行编程 ”) :

1. 1. A : 加载命令 “0100 0000”
2. 2. C : 加载数据低字节。若某一位为 "0” 表示需要进行编程 , 否则需要擦除
3. 3. 将 BS1 置 "1”、 BS2 置 "1” , 选择扩展数据字节
4. 4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高
5. 5. 将 BS2 置 "0” , 选择低位字节

Figure 123. 熔丝位编程波形



锁定位编程

锁定位编程步骤如下 (命令及数据装在细节见 P263” 对 Flash 进行编程 ”) :

1. A : 加载命令 “0010 0000”
2. C.:加载数据低字节 ,位 n 为 "0” 表示此锁定位需要编程。如果芯片已经处于 LB 模式 3(LB1 及 LB2 被编程) , 那么就不可能通过外部编程命令对锁定位进行编程
3. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高

锁定位只能通过芯片擦除命令来清除。

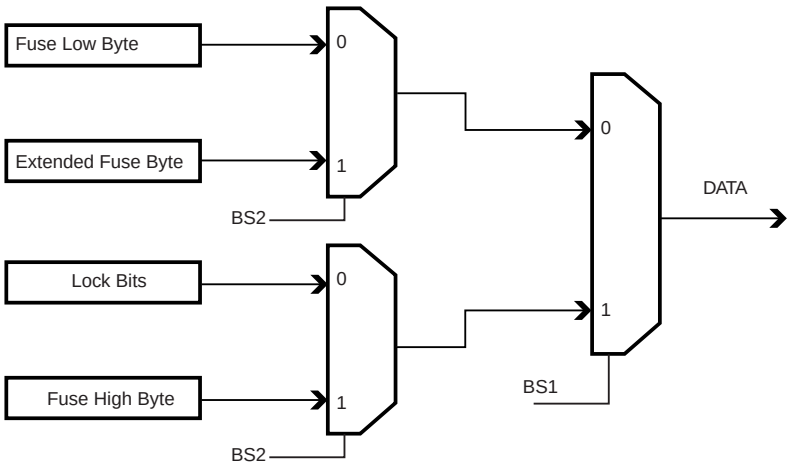
读取熔丝位及锁定位

读取熔丝位及锁定位的步骤如下 (命令加载细节见 P263” 对 Flash 进行编程 ”) :

1. A : 加载命令 “0000 0100”
2. 将 $\overline{OE}$、 BS2 和 BS1 置 "0” , 然后从 DATA 读取熔丝低位的状态 ("0” 表示已编程)
3. 将$\overline{OE}$置"0” ,BS2和BS1置“1” ,然后从DATA读取熔丝高位的状态("0”表示已编程)

4. 将 $\overline{OE}$ 和 BS1 置 "0" , BS2 置 “1” , 然后从 DATA 读取扩展熔丝位的状态 ("0" 表示已编程)
5. 将 $\overline{OE}$ 置 "0" , BS2 置 "0" , BS1 置 “1” , 然后从 DATA 读取锁定位的状态 ("0" 表示已编程)
6. 将 $\overline{OE}$ 置 “1”。

Figure 124. 读操作过程中 BS1、 BS2 与熔丝位及锁定位的对应关系



读取标识字节

读取标识字节的算法如下 (命令及地址加载参考 P263” 对 Flash 进行编程 ”) :

1. A : 加载命令 “0000 1000”
2. B : 加载地址低字节 0x00 - 0x02
3. 将 $\overline{OE}$ 置 "0” , BS1 置 "1” , 然后从 DATA 读取标识字节
4. 将 $\overline{OE}$ 置 "1”

读取校准字节

读取校准字节的算法如下 (命令及地址加载参考 P263” 对 Flash 进行编程 ”) :

1. A : 加载命令 “0000 1000”
2. B : 加载地址低字节 0x00
3. 将 $\overline{OE}$ 置 "0” , BS1 置 “1” , 然后从 DATA 读取校准字节
4. 将 $\overline{OE}$ 置 “1”

并行编程特性

Figure 125. 并行编程时序 , 包括一些常规的时序要求

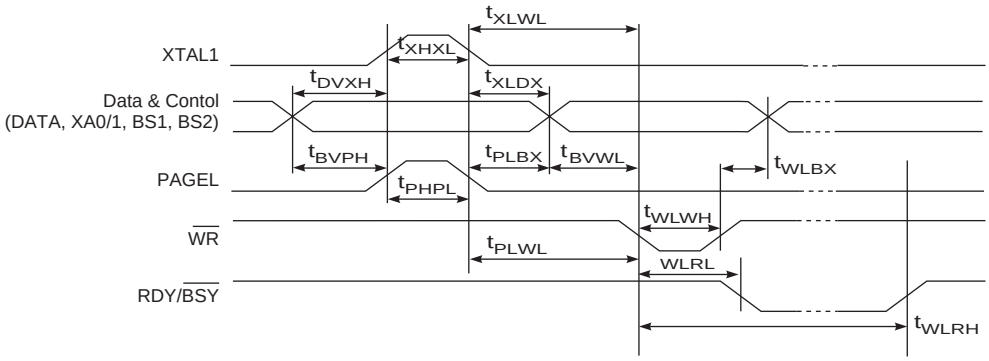
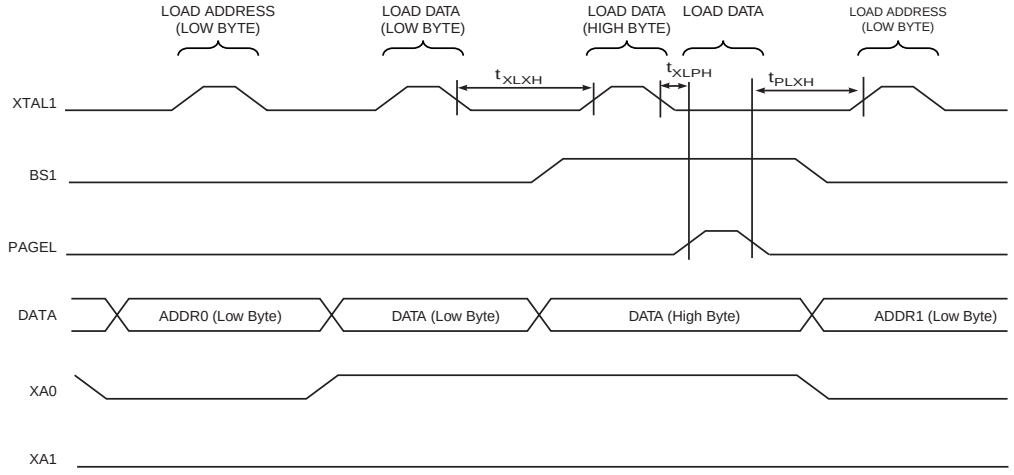
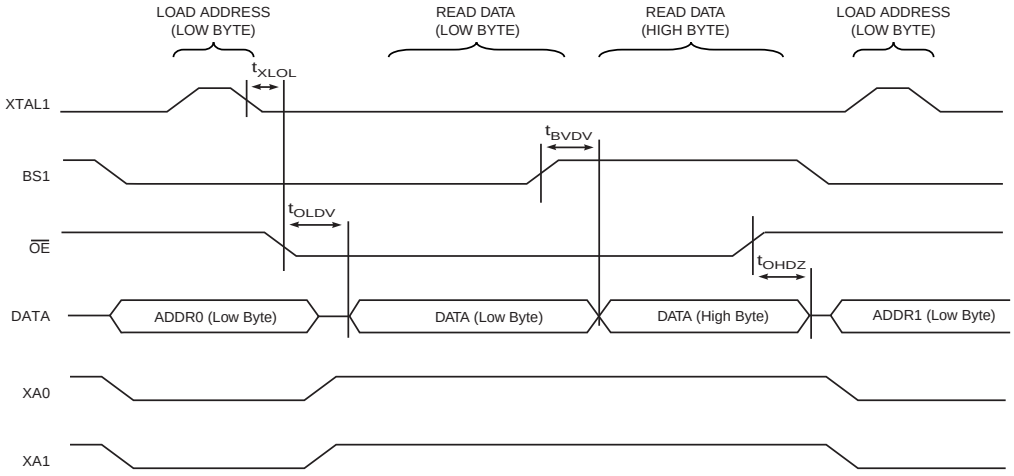


Figure 126. 并行编程时序，有时序要求的加载序列 (1)



Note: 1. Figure125 给出的时序要求 (t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于加载操作。

Figure 127. 并行编程时序，有时序要求的读序列 (同一页)(1)



Note: 1. Figure125 给出的时序要求 (即 t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于读操作。

Table 129. 并行编程参数 V_{CC} = 5V ± 10%

符号	参数	最小值	典型值	最大值	单位
V_{PP}	编程使能电压	11.5		12.5	V
I_{PP}	编程使能电流			250	µA
t_{DVXH}	在 XTAL1 为高之前数据及控制有效	67			ns
t_{XLXH}	从 XTAL1 低到 XTAL1 高	200			ns
t_{XHXL}	XTAL1 为高时的脉宽	150			ns
t_{XLDX}	XTAL1 为低之后数据及控制保持	67			ns
t_{XLWL}	从 XTAL1 低到 $\overline{WR}$ 低	0			ns
t_{XLPH}	从 XTAL1 低到 PAGEL 高	0			ns

Table 129. 并行编程参数 V_{CC} = 5V ± 10% (Continued)

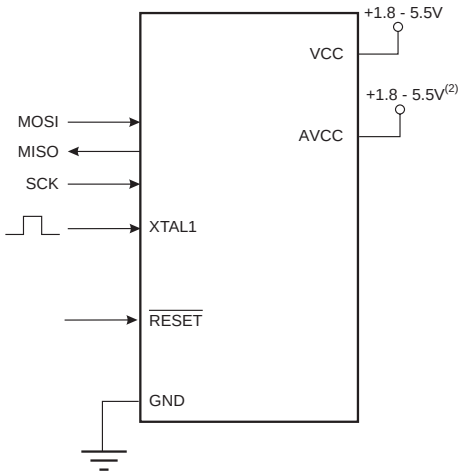
符号	参数	最小值	典型值	最大值	单位
t_{PLXH}	从 PAGEL 低到 XTAL1 高	150			ns
t_{BVPH}	PAGEL 为高之前 BS1 有效	67			ns
t_{PHPL}	PAGEL 为高时的脉宽	150			ns
t_{PLBX}	PAGEL 为低之后 BS1 保持	67			ns
t_{WLBX}	$\overline{WR}$ 为低之后 BS2/1 保持	67			ns
t_{PLWL}	从 PAGEL 低到 $\overline{WR}$ 为低	67			ns
t_{BVWL}	BS1 有效至 $\overline{WR}$ 为低	67			ns
t_{WLWH}	$\overline{WR}$ 为低时的脉宽	150			ns
t_{WLRL}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为低	0		1	µs
t_{WLRH}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为高 (1)	3.7		4.5	ms
t_{WLRH_CE}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为高，芯片擦除操作 (2)	7.5		9	ms
t_{XLOL}	从 XTAL1 低到 $\overline{OE}$ 为低	0			ns
t_{BVDV}	BS1 有效至 DATA 有效	0		250	ns
t_{OLDV}	从 $\overline{OE}$ 低到 DATA 有效			250	ns
t_{OHDZ}	从 $\overline{OE}$ 低到 DATA 为三态			250	ns

Notes: 1. 在进行 Flash、 EEPROM、熔丝位及锁定位写操作时 t_{WLRH} 有效。
2. 在执行芯片擦除操作时 t_{WLRH_CE} 有效。

串行下载

当 $\overline{RESET}$ 位低电平时，可以通过串行 SPI 总线对 Flash 及 EEPROM 进行编程。串行接口包括 SCK、 MOSI(输入) 及 MISO(输出)。 $\overline{RESET}$ 为低之后，应在执行编程 / 擦除操作之前执行编程允许指令。P261Table128 列出了 SPI 编程所需引脚的映射。不是所有的器件都使用 SPI 引脚专用于内部 SPI 接口。

Figure 128. 串行编程及校验 (1)



Notes: 1. 如果芯片由片内振荡器提供时钟，那么就不用在 XTAL1 引脚上连接时钟源。
2. V_{CC} - 0.3V < AVCC < V_{CC} + 0.3V，但是 AVCC 必须在 1.8 - 5.5V 范围内。

编程 EEPROM 时， MCU 在自定时的编程操作中会插入一个自动擦除周期，从而无需执行芯片擦除命令。芯片擦除操作将程序存储器及 EEPROM 的内容都擦除为 0xFF。

时钟通过 CKSEL 熔丝位确定。串行时钟 (SCK) 的最小低电平时间和最小高电平时间要满足如下要求：

低 :> f_{ck} < 12 MHz 时为 2 个 CPU 时钟周期，f_{ck} >= 12 MHz 时为 3 个 CPU 时钟周期

高 :> f_{ck} < 12 MHz 时为 2 个 CPU 时钟周期，f_{ck} >= 12 MHz 时为 3 个 CPU 时钟周期

串行编程算法

向 ATmega48/88/168 串行写入数据时，数据在 SCK 的上升沿得以锁存。

从 ATmega48/88/168 读取数据时，数据在 SCK 的下降沿输出。时序细节见 Figure129 。

在串行编程模式下对 ATmega48/88/168 进行编程及校验时，应遵循以下的步骤（见 Table131 中的 4 字节指令格式）：

1. 上电顺序：
   在 RESET 及 SCK 为 "0" 时，向 V_{CC} 及 GND 供电。在一些系统中，编程器不能保证在上电时 SCK 保持为低。在这种情况下， SCK 拉低之后应在 RESET 加一正脉冲，而且这个脉冲至少要维持 2 个 CPU 时钟周期。
2. 上电之后等待至少 20 ms，然后向 MOSI 引脚输入串行编程使能指令以使能串行编程。
3. 通信不同步将造成串行编程指令不工作。同步之后，在发送编程使能指令的第三个字节时，第二个字节的内容 (0x53) 将被反馈回来。不论反馈的内容正确与否，指令的 4 个字节必须全部传输。如果 0x53 未被反馈，则需要向 RESET 提供一个正脉冲以开始新的编程使能指令。
4. Flash 的编程以一次一页的方式进行。在执行加载程序存储页指令时，通过 6 LSB 的地址信息，数据以字节为单位加载到存储页。为保证加载的正确性，应先向给定地址传送数据低字节，之后是高字节。程序存储页通过地址的高 8 位以及写程序存储器页指令获得数据。如果不使用查询的方式，那么在操作下一页数据之前应等待至少 t_{WD_FLASH} 的时间（见 Table130.）。在 Flash 写操作完成之前访问串行编程接口会导致编程错误。
5. 提供了地址及数据信息之后，适合的写指令将以字节为单位对 EEPROM 编程。EEPROM 存储单元总是在写入新数据之前自动擦除。如果不使用查询的方式，那么在操作下一页数据之前应等待至少 t_{WD_EEPROM} 的时间（见 Table130.）。对于全片擦除之后的芯片，数据为 0xFF 的不需要编程。
6. 可通过读指令来校验任何一个存储单元的内容。数据从串行输出口 MISO 输出。
7. 编程结束后可以将 RESET 拉高开始正常操作。
8. 下电序列（如果需要）：
   将 RESET 置 “1”。
   切断 V_{CC}。

Flash 数据查询

当 Flash 正处于某一页的编程状态时，读取此页中的内容将得到 0xFF。编程结束后，被编程的数据即可以正确读出。通过这种方法可以确定何时可以写下一页。由于整个页是同时编程的，这一页中的任何一个地址都可以用来查询。 Flash 数据查询不适用于数据 0xFF。因此，在编程 0xFF 时，用户至少要等待 t_{WD_FLASH} 才能进行下一页的编程。由于全片擦除将所有的单元擦为 0xFF，所以编程数据为 0xFF 时可以跳过这个操作。 t_{WD_FLASH} 的值见 Table130 。

EEPROM 数据查询

当 EEPROM 正在处理一个字节的编程操作时，读取此地址将返回 0xFF。编程结束后，被编程的数据即可以正确读出。这一方法可用来判断何时可以写下一个字节。数据查询对数据 0xFF 无效。但用户应该考虑到，全片擦除将所有的单元擦为 0xFF，所以编程数据为 0xFF 时可以跳过这个操作。不过这不适用于全片擦除时 EEPROM 内容被保留的情况。用户若在此时编程 0xFF，在进行下一字节编程之前至少等待 t_{WD_EEPROM} 的时间。t_{WD_EEPROM} 的值见 Table130。

Table 130. 写下一个 Flash 或 EEPROM 单元之前的最小等待时间

符号	最小等待时间
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	3.6 ms
t_{WD_ERASE}	9.0 ms

Figure 129. 串行编程波形图

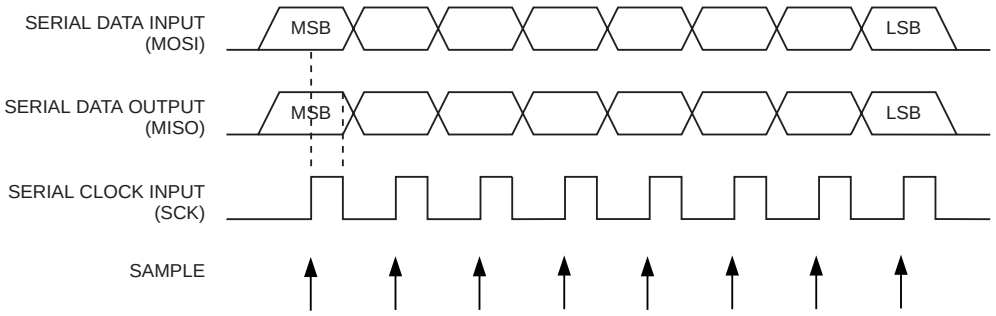


Table 131. 串行编程指令集

指令	指令格式				操作
	字节 1	字节 2	字节 3	字节 4	
编程使能	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	$\overline{\text{RESET}}$ 拉低后使能串行编程
全片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	擦除 EEPROM 及 Flash
读程序存储器	0010 **H**000	000**a aaaa**	**bbbb bbbb**	**oooo oooo**	从字地址为 **a**:**b** 的程序存储器读取 **H**(高或低字节) 数据的 **o**
加载程序存储器页	0100 **H**000	000x xxxx	xx**bb bbbb**	**iiii iiii**	向字地址为 **b** 的程序存储页 **H**(高或低字节) 写入数据 **i**。应先写低字节再写高字节
写程序存储器页	0100 1100	000**a aaaa**	**bb**xx xxxx	xxxx xxxx	在地址 **a**:**b** 加载程序存储页
读 EEPROM 存储器	1010 0000	000x xx**aa**	**bbbb bbbb**	**oooo oooo**	从 EEPROM 的地址 **a**:**b** 处读出数据 **o**
写 EEPROM 存储器	1100 0000	000x xx**aa**	**bbbb bbbb**	**iiii iiii**	向 EEPROM 地址 **a**:**b** 处中写入数据 **o**
加载 EEPROM 存储器页 (页寻址)	1100 0001	0000 0000	0000 00**bb**	**iiii iiii**	将数据 **i** 加载到 EEPROM 存储器页缓冲区。数据加载完毕后对 EEPROM 页进行编程
写EEPROM存储器页(页寻址)	1100 0010	00xx xx**aa**	**bbbb bb**00	xxxx xxxx	对地址为 **a**:**b** 的 EEPROM 执行页写操作
读锁定位	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	读锁定位。 ”0” 为已编程 ， "1” 为未编程。细节见 P254Table115
写锁定位	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	写锁定位。写 “0” 表示编程锁定位。细节见 P254Table115
读标识字节	0011 0000	000x xxxx	xxxx xx**bb**	**oooo oooo**	从地址 **b** 读取标识字节 **o**
写熔丝位	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	“0” 表示已编程 ， “1” 表示未编程。见 **Table XXX on page XXX**
写高熔丝位	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	“0” 表示已编程 , “1” 表示未编程。见 P221Table98
写扩展熔丝位	1010 1100	1010 0100	xxxx xxxx	**xxxx xxii**	“0” 表示已编程 ， “1” 表示未编程。见 P256Table118
读熔丝位	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	Read Fuse bits. “0” = programmed, “1” = unprogrammed. See 读熔丝位。 “0” 表示已编程 ， “1” 表示未编程。细节见 **Table XXX on page XXX**

Table 131. 串行编程指令集 (Continued)

指令	指令格式				操作
	字节 1	字节 2	字节 3	字节 4	
读高熔丝位	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	读熔丝高位。“0” 表示已编程，“1” 表示未编程。细节见 P221Table98
读扩展熔丝位	0101 0000	0000 1000	xxxx xxxx	**oooo oooo**	读扩展熔丝位。“0” 表示已编程，“1” 表示未编程。细节见 P256Table118
读校准字节	0011 1000	000x xxxx	0000 0000	**oooo oooo**	读校准字节
查询 RDY/$\overline{BSY}$	1111 0000	0000 0000	xxxx xxxx	xxxx xxx**o**	**o** = “1” 表示编程操作正在进行。等到它为 “0” 后可以执行其他命令。

Note: **a** = 地址高位， **b** = 地址低位， **H** = 0 - 低字节， 1 - 高字节 ，**o** = 数据输出，**i** = 数据输入，x = 任意值

SPI 串行编程特性参数 SPI 模块的特性参数请见 P280 “SPI 时序特性 ” 。

电气特性

绝对极限值 *

工作温度	-55°C 到 +125°C
存储温度	-65°C 到 +150°C
各个引脚对地的电压，除了 $\overline{\text{RESET}}$	-0.5V 到 V_{CC}+0.5V
$\overline{\text{RESET}}$ 引脚对地的电压	-0.5V 到 +13.0V
最大工作电压	6.0V
每个 I/O 引脚上的直流电流	40.0 mA
V_{CC} 与 GND 引脚上的直流电流	200.0 mA

*NOTICE: 如果强制芯片在超出“ 绝对极限值” 表中所列的条件之下工作可能造成器件的永久损坏。这仅是工作应力的极限。并不表示器件可以工作于表中所列条件之下，或是那些超越工作范围明确规定的其他条件之下。长时间工作于绝对极限值可能会影响器件的寿命。

直流特性

T_A = -40°C-85°C, V_{CC} = 1.8V-5.5V (除非另外说明)

符号	参数	条件	最小值	典型值	最大值	单位
V_{IL}	输入低电压，除了 XTAL1 及 RESET 引脚	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IL1}	输入低电压， XTAL1 引脚	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IL2}	输入低电压， RESET 引脚	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IH}	输入高电压，除了 XTAL1 及 RESET 引脚	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IH1}	输入高电压， XTAL1 引脚	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$(2) $0.7V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IH2}	输入高电压， RESET 引脚	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$(2)		V_{CC} + 0.5	V
V_{OL}	输出低电压 (3)，除 PC6	I_{OL} = 10mA, V_{CC} = 5V I_{OL} = 5mA, V_{CC} = 3V			0.7 0.5	V
V_{OL1}	输出低电压 (3)，PC6	TBD			TBD	V
V_{OH}	Output High Voltage(4), Except PC6	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.3			V
V_{OH1}	输出高电压 (4)，除 PC6	TBD	TBD			V
I_{IL}	输入漏电流， I/O 引脚	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	输入漏电流， I/O 引脚	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset 引脚上拉电阻		30		60	kΩ
R_{PU}	I/O 引脚上拉电阻		20		50	kΩ

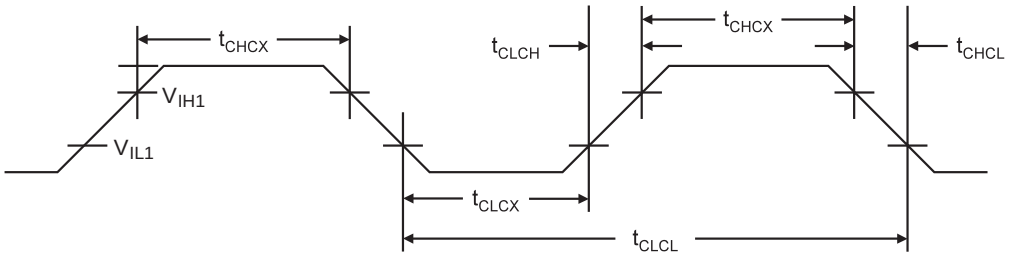
T_A = -40°C-85°C, V_{CC} = 1.8V-5.5V (除非另外说明) (Continued)

符号	参数	条件	最小值	典型值	最大值	单位
I_{CC}	电源电流	工作于 1MHz，V_{CC} = 2V (ATmega48/88/168V)			0.55	mA
		工作于 4MHz，V_{CC} = 3V (ATmega48/88/168L)			3.5	mA
		工作于 8MHz，V_{CC} = 5V (ATmega48/88/168)			12	mA
		空闲 1MHz， V_{CC} = 2V (ATmega48/88/168V)		0.25	0.5	mA
		空闲 4MHz，V_{CC} = 3V (ATmega48/88/168L)			1.5	mA
		空闲 8MHz，V_{CC} = 5V (ATmega48/88/168)			5.5	mA
	掉电模式	WDT 使能， V_{CC} = 3V		<8	15	µA
		WDT 禁用 V_{CC} = 3V		<1	2	µA
V_{ACIO}	模拟比较器输入偏置电压	V_{CC} = 5V V_{in} = V_{CC}/2		<10	40	mV
I_{ACLK}	模拟比较器输入漏电流	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	模拟比较器传输延时	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Notes:
1. “ 最大值” 表示保证引脚读取数值为低时的最高值
2. “ 最小值” 表示保证引脚读取数值为高时的最低值
3. 虽然在稳定状态条件(非瞬态)下每个I/O端口都可以吸收比测试条件下更多的电流(20 mA ,V_{CC} = 5V；10 mA，V_{CC} = 3V) ，但是仍需要遵循以下要求：
   ATmega48：
   1] 端口 C0 - C5 的 IOL 总和不能超过 70 mA
   2] 端口 C6、 D0 - D4 的 IOL 总和不能超过 70 mA
   3] 端口 B0 - B7、 D5 - D7 的 IOL 总和不能超过 70 mA
   ATmega88/168：
   1] 端口 C0 - C5 的 IOL 总和不能超过 100 mA
   2] 端口 C6、D0 - D4 的 IOL 总和不能超过 100 mA
   3] 端口 B0 - B7、D5 - D7 的 IOL 总和不能超过 100 mA
   如果 IOL 超过了测试条件， VOL 可能超过相关指标。不保证引脚可以吸收比列于此处的测试条件更大的电流。
4. 虽然在稳定状态条件 (非瞬态) 下每个 I/O 端口都可以输出比测试条件下更多的电流 (20 mA ,V_{CC} = 5V ； 10 mA，V_{CC} = 3V) ，但是需要遵循以下要求：
   ATmega48：
   1] 端口 C0 - C5 的 IOH 总和不能超过 70 mA
   2] 端口 C6、 D0 - D4 的 IOH 总和不能超过 70 mA
   3] 端口 B0 - B7、 D5 - D7 的 IOH 总和不能超过 70 mA.
   ATmega88/168：
   1] 端口 C0 - C5 的 IOH 总和不能超过 100 mA
   2] 端口 C6、D0 - D4 的 IOH 总和不能超过 100 mA
   3] 端口 B0 - B7、D5 - D7 的 IOH 总和不能超过 100 mA.
   如果 IOH 超过了测试条件， VOH 可能超过相关指标。不保证引脚可以输出比列于此处的测试条件更大的电流。
5. 本手册中所有的直流特性数据与其它 AVR 微控制器手册一样，均是基于仿真的，这些值仅是设计目标，实际值在对实际芯片的测试后会更新。

外部时钟波形

Figure 130. 外部时钟波形



外部时钟驱动

Table 132. 外部时钟驱动

符号	参数	V_{CC}=1.8-5.5V		V_{CC}=2.7-5.5V		V_{CC}=4.5-5.5V		单位
		最小值	最大值	最小值	最大值	最小值	最大值	
$1/t_{CLCL}$	振荡器频率	0	2	0	8	0	16	MHz
t_{CLCL}	时钟周期	500		125		62.5		ns
t_{CHCX}	高电平时间	200		50		25		ns
t_{CLCX}	低电平时间	200		50		25		ns
t_{CLCH}	上升时间		2.0		1.6		0.5	µs
t_{CHCL}	下降时间		2.0		1.6		0.5	µs
Δt_{CLCL}	时钟周期的变化		2		2		2	%

Note: 本手册中所有的直流特性数据与其它 AVR 微控制器手册一样，均是基于仿真的，这些值仅是设计目标，实际值在对实际芯片的测试后会更新。

Figure 131. ATmega48V/88V/168V 中最高频率与 V_{CC} 的关系

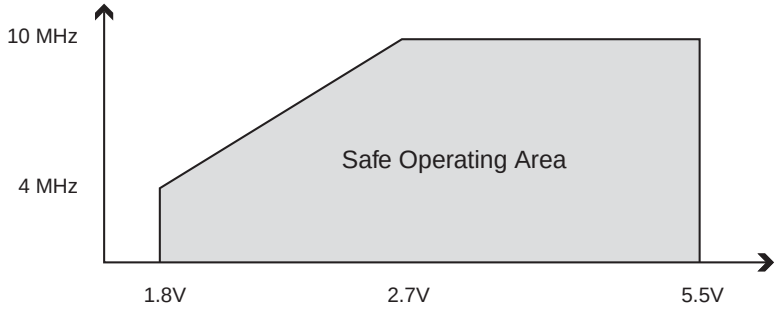
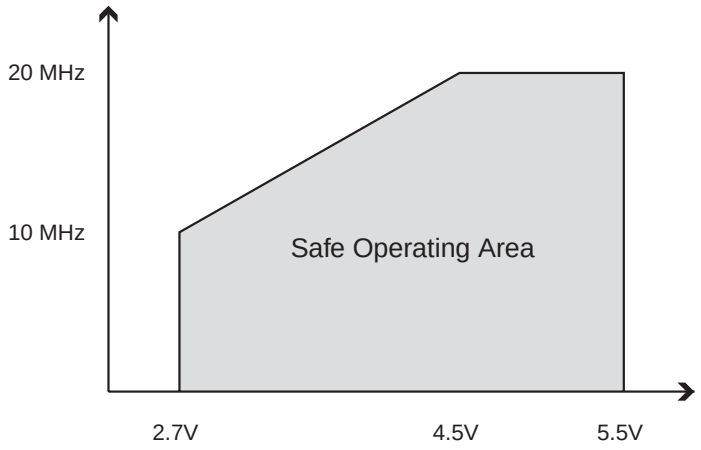


Figure 132. ATmega48/88/168 中最高频率与 V_{CC} 的关系



两线串行接口特性

Table133 给出器件与两线串行总线连接的要求。ATmega48/88/168 两线串行接口达到或超出下面提到的条件。时序符号参见 Figure133。

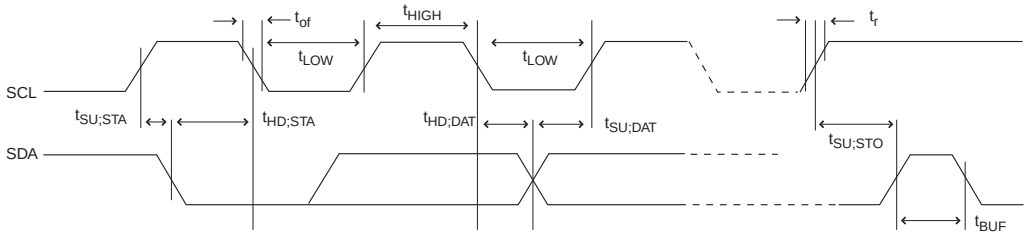
Table 133. 两线串行总线要求

符号	参数	条件	最小值	最大值	单位
V_{IL}	输入低电压		-0.5	0.3 V_{CC}	V
V_{IH}	输入高电压		0.7 V_{CC}	V_{CC} + 0.5	V
V_{hys}(1)	施密特迟滞触发输入		0.05 V_{CC}(2)	–	V
V_{OL}(1)	输出低电压	3 mA 反向电流	0	0.4	V
t_r(1)	SDA 与 SCL 上升时间		20 + 0.1C_b(3)(2)	300	ns
t_{of}(1)	从 V_{IHmin} 到 V_{ILmax} 输出下降时间	10 pF < C_b < 400 pF(3)	20 + 0.1C_b(3)(2)	250	ns
t_{SP}(1)	通过输入滤波器抑制尖峰		0	50(2)	ns
I_i	每个 I/O 引脚的输入电流	0.1V_{CC} < V_i < 0.9V_{CC}	-10	10	µA
C_i(1)	每个 I/O 引脚的电容		–	10	pF
f_{SCL}	SCL 时钟频率	f_{CK}(4) > max(16f_{SCL}, 250kHz)(5)	0	400	kHz
Rp	上拉电阻值	f_{SCL} ≤ 100 kHz	$\frac{V_{CC}-0,4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
		f_{SCL} > 100 kHz	$\frac{V_{CC}-0,4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
$t_{HD;STA}$	(repeated) START 信号保持时间	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{LOW}	SCL 时钟低周期	f_{SCL} ≤ 100 kHz(6)	4.7	–	µs
		f_{SCL} > 100 kHz(7)	1.3	–	µs
t_{HIGH}	SCL 时钟高周期	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{SU;STA}$	repeated START 信号启动时间	f_{SCL} ≤ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{HD;DAT}$	数据保持时间	f_{SCL} ≤ 100 kHz	0	3.45	µs
		f_{SCL} > 100 kHz	0	0.9	µs
$t_{SU;DAT}$	数据启动时间	f_{SCL} ≤ 100 kHz	250	–	ns
		f_{SCL} > 100 kHz	100	–	ns
$t_{SU;STO}$	STOP 信号启动时间	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{BUF}	STOP 与 START 信号间总线闲时间	f_{SCL} ≤ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	1.3	–	µs

Notes:
1. ATmega48/88/168 中，该参数仅为表征值，并未 100% 测试
2. 只要求 f_{SCL} > 100 kHz
3. C_b 为单根总线上的电容值，单位 pF
4. f_{CK} 为 CPU 时钟频率

5. 以上要求适合 ATmega48/88/168 两线串行接口操作。其它器件与两线串行总线的连接仅需满足 f_{SCL} 要求
6. 实际由 ATmega48/88/168 产生的两线串行接口低周期为 ($1/f_{SCL} - 2/f_{CK}$)，因此为满足 f_{SCL} = 100 kHz 的要求， f_{CK} 必须大于 6 MHz
7. 实际由 ATmega48/88/168 产生的两线串行接口低周期为 ($1/f_{SCL} - 2/f_{CK}$)，因此当 f_{CK} = 8 MHz 时，低时间要求不是严格达到 f_{SCL} > 308 kHz。到目前为止，ATmega48/88/168 器件间全速通信 (400 kHz)，要求对方有适当的 t_{LOW} 接收容限。

Figure 133. 两线串行总线时序图



SPI 时序特性

细节见 Figure134 及 Figure135。

Table 134. SPI 时序参数

	说明	模式	最小值	典型值	最大值	
1	SCK 周期	主机		见 Table71		ns
2	SCK 高 / 低电平	主机		50% 占空比		
3	上升 / 下降时间	主机		3.6		
4	建立时间	主机		10		
5	保持时间	主机		10		
6	输出到 SCK	主机		$0.5 \cdot t_{sck}$		
7	SCK 到输出	主机		10		
8	SCK 到输出高电平	主机		10		
9	$\overline{SS}$ 低到输出	从机		15		
10	SCK 周期	从机	$4 \cdot t_{ck}$			
11	SCK 高 / 低电平 (1)	从机	$2 \cdot t_{ck}$			
12	上升 / 下降时间	从机		1600		
13	启动	从机	10			
14	保持时间	从机	t_{ck}			
15	SCK 到输出	从机		15		
16	SCK 到 $\overline{SS}$ 高	从机	20			
17	$\overline{SS}$ 高到三态	从机		10		
18	$\overline{SS}$ 低到 SCK	从机	20			

Note:
1. 在 SPI 编程模式，最小的 SCK 高 / 低电平时间为：
   f_{CK} < 12 MHz 下 - 2 t_{CLCL}
   f_{CK} > 12 MHz 下 - 3 t_{CLCL}
2. 本手册中所有的直流特性数据与其它 AVR 微控制器手册一样，均是基于仿真的，这些值仅是设计目标，实际值在对实际芯片的测试后会更新。

Figure 134. SPI 接口时序要求 (主机模式)

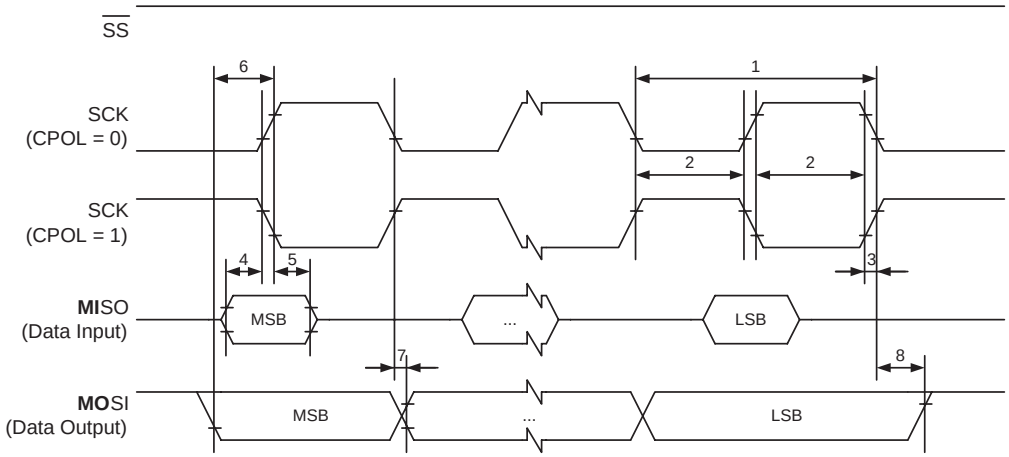
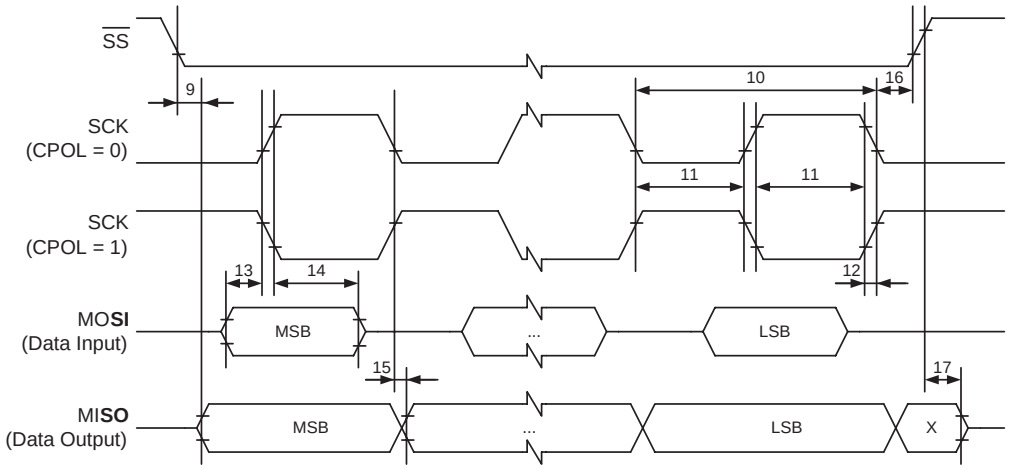


Figure 135. SPI 接口时序模式 (从机模式)



ADC 特性—初始参数

Table 135. ADC 特性参数

符号	参数	条件	最小值	典型值	最大值	单位
	分辨率			10		Bits
	绝对精度 (包括 INL、 DNL、量化误差、增益及偏置误差)	V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		2	2.5	LSB
		V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 1 MHz		4.5		LSB
		V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz 噪声抑制模式		2		LSB
		V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 1 MHz 噪声抑制模式		4.5		LSB
	整体非线性 (INL)	V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		0.5		LSB
	差分非线性 (DNL)	V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		0.25		LSB
	增益误差	V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		2		LSB
	偏移误差	V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		2		LSB
	转换时间	连续转换	13		260	µs
	时钟频率		50		1000	kHz
AVCC	模拟电压		V_{CC} - 0.3		V_{CC} + 0.3	V
V_{REF}	参考电压		1.0		AVCC	V
V_{IN}	输入电压		GND		V_{REF}	V
	输入带宽			38.5		kHz
V_{INT}	片内参考电压		1.0	1.1	1.2	V
R_{REF}	参考电压输入阻抗			32		kΩ
R_{AIN}	模拟输入阻抗			100		MΩ

Note: 本数据手册中所有的直流特性数据均来源于对器件的仿真 ,以及其他基于相同生产工艺的 AVR 为控制器的标定特性。这些值为初始数据，代表的是设计目标，在对实际芯片进行特性化之后将得到更新。

ATmega48/88/168 典型特性 – 初始数据

本数据手册所有的直流特性数据均来源于对器件的仿真，以及其他基于相同生产工艺的 AVR 为控制器的标定特性。这些值为初始数据，代表的是设计目标，在对实际芯片进行特性化之后将得到更新。

下面的图表给出了器件的典型性能。在产生过程中并不测试这些数据。全部电流测量数据都是在所有的 I/O 引脚配置为输入且内部上拉电阻使能的条件下测得的。时钟源为外部正弦波发生器产生的满幅值正弦波。

掉电模式下的功耗与选择的时钟无关。

耗电流与多个因素有关：工作电压、工作频率、 I/O 的负载、 I/O 引脚开关速率、执行的代码及环境温度。最主要的因素是工作电压和工作频率。

容性负载 I/O 的输出电流可通过公式 C_L*V_{CC}*f 进行估算， C_L = 负载电容， V_{CC} = 工作电压， f = I/O 引脚平均开关频率。

器件的特性化是在比测试极限频率更高的频率进行的。但是不保证器件能够正常在高于订货信息表给出的工作频率。

看门狗使能的掉电模式和看门狗禁止的掉电模式之间的电流差值即为开关看门狗定时器所需的电流。

工作电流

Figure 136. 工作电流和工作频率 (0.1 - 1.0 MHz) 的关系

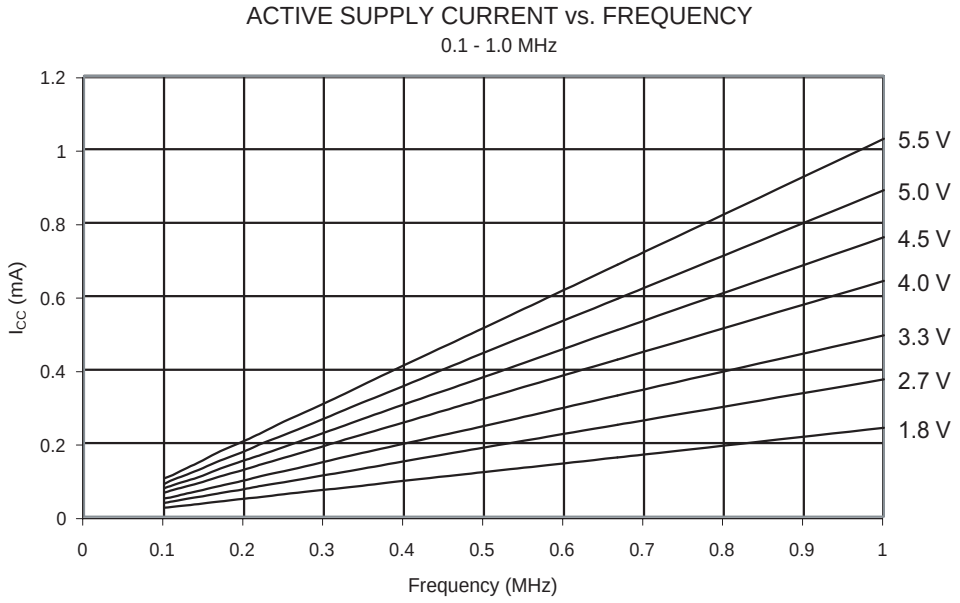


Figure 137. 工作电流和工作频率 (1 - 20 MHz) 的关系

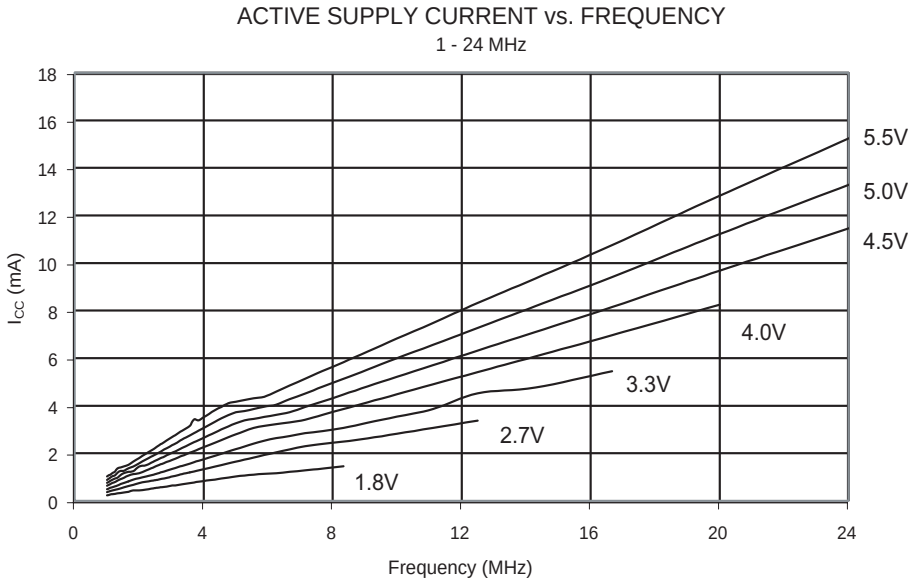


Figure 138. 工作电流和 V_{CC} 的关系 (内部 RC 振荡器，8 MHz)

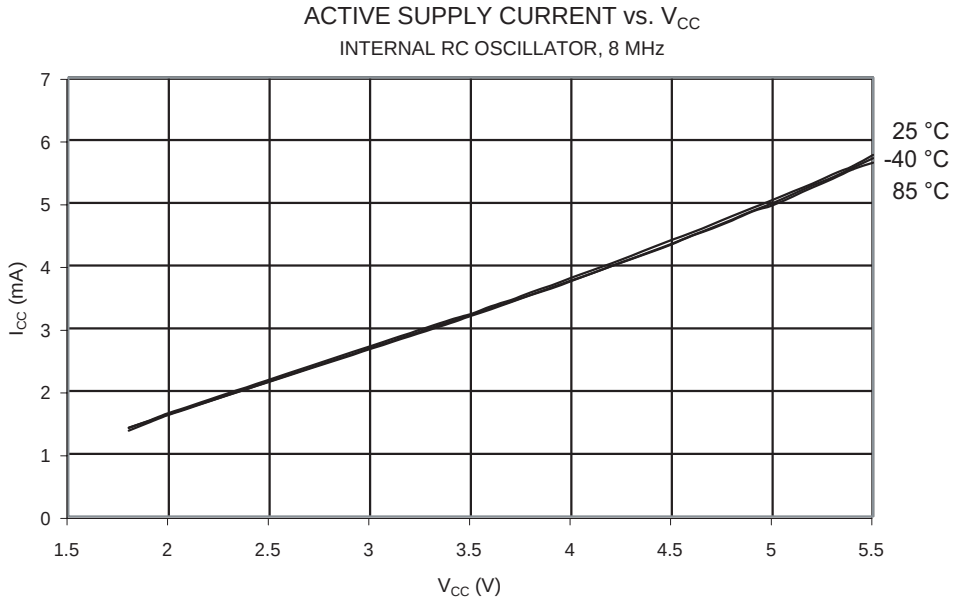


Figure 139. 工作电流和 V_{CC} 的关系 (内部 RC 振荡器 ， CKDIV8 编程， 1 MHz)

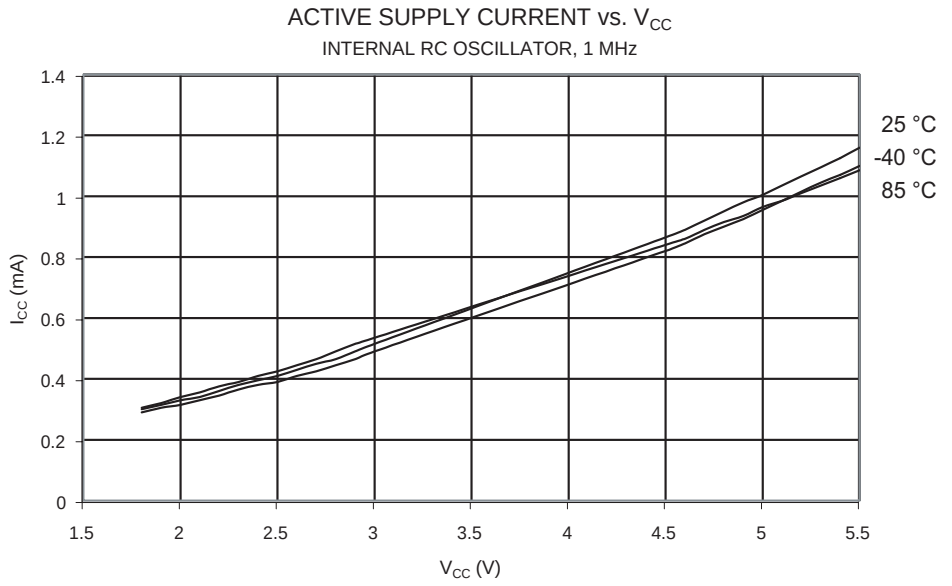
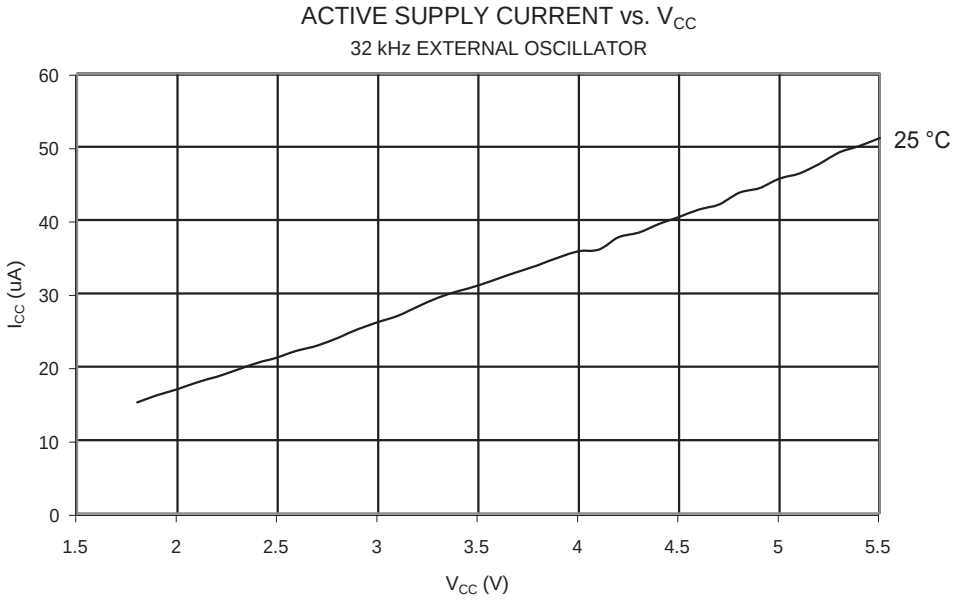


Figure 140. 工作电流和 V_{CC} 的关系 (32 kHz 外部晶振)



空闲模式电流

Figure 141. 空闲模式电流和工作频率 (0.1 - 1.0 MHz) 的关系

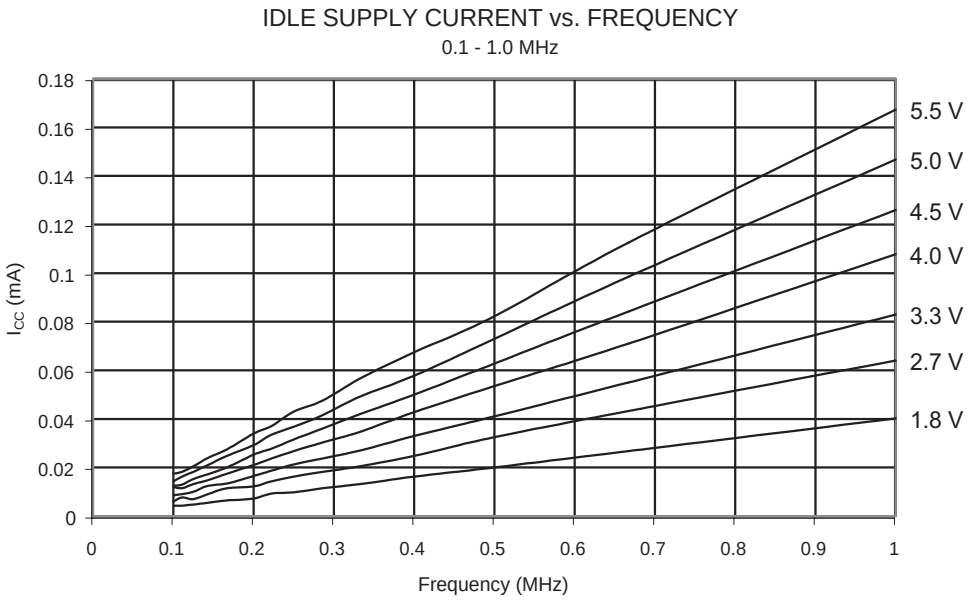


Figure 142. 空闲模式电流和工作频率 (1 - 20 MHz) 的关系

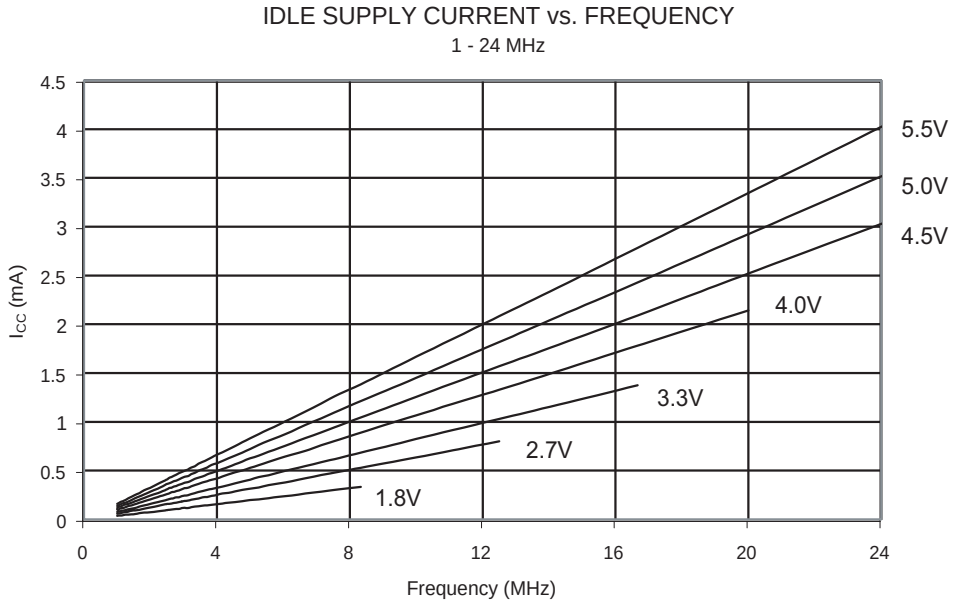


Figure 143. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， 8 MHz)

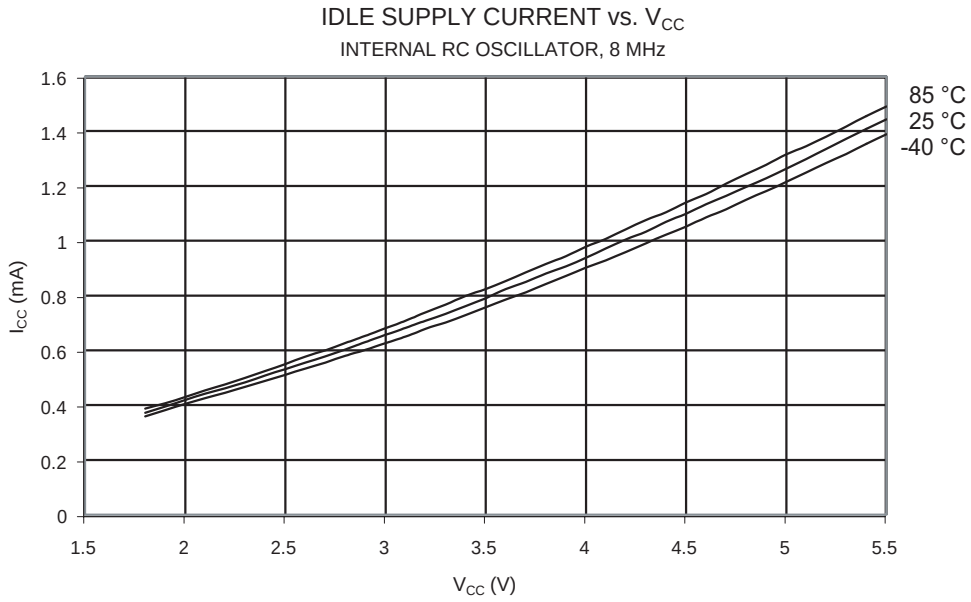


Figure 144. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， CKDIV8 编程，1 MHz)

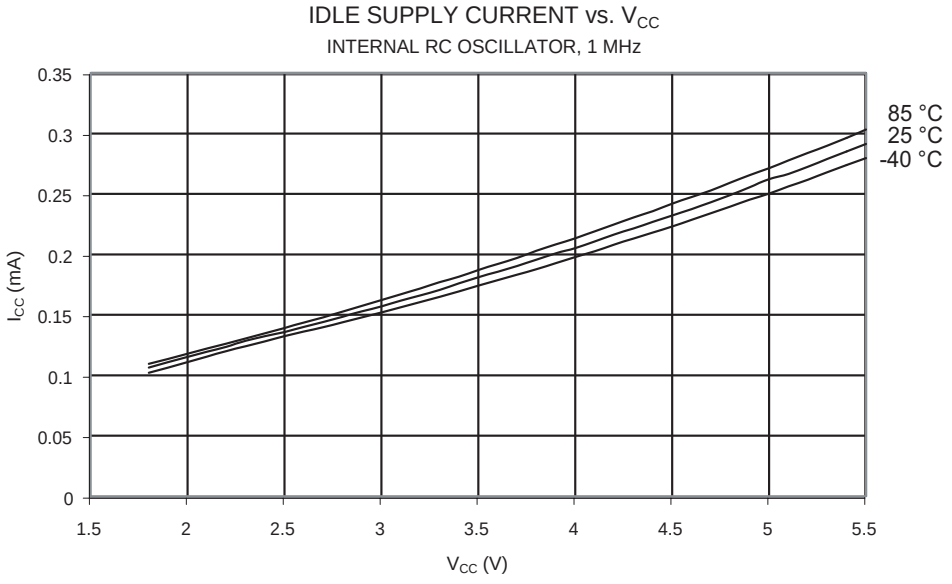
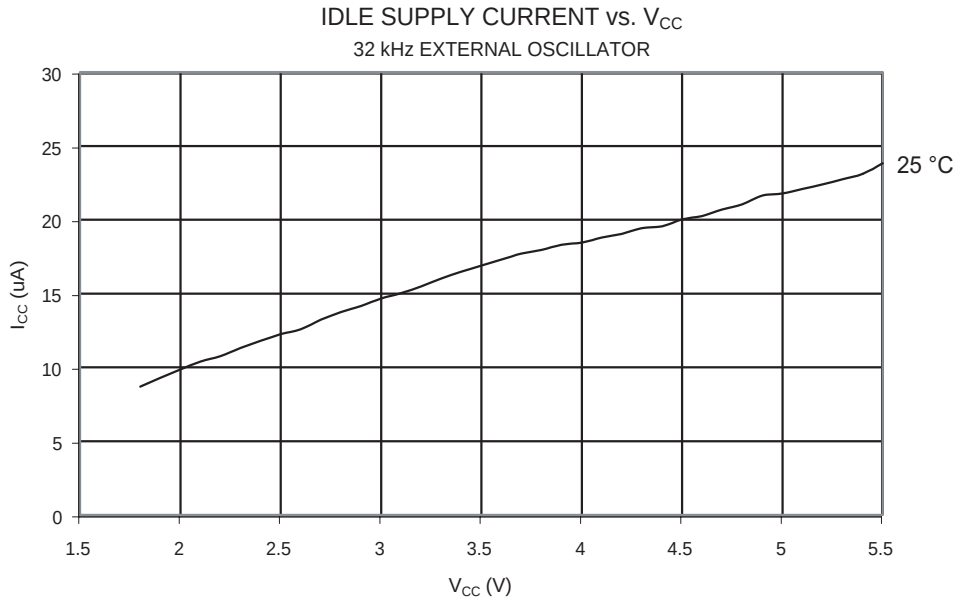


Figure 145. 空闲模式电流和 V_{CC} 的关系 (32 kHz 外部晶振)



掉电模式电流

Figure 146. 掉电模式电流和 V_{CC} 的关系 (看门狗定时器禁用)

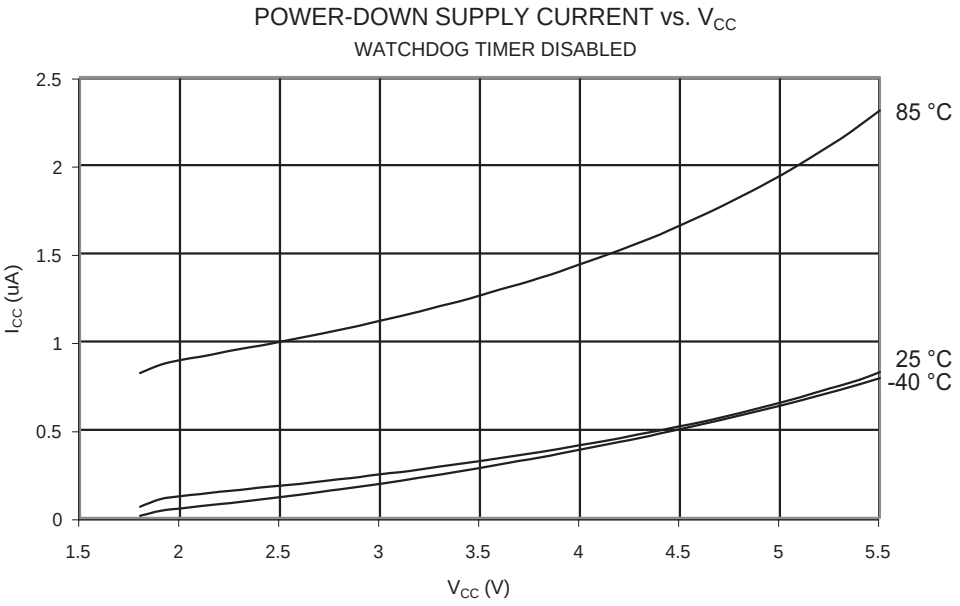
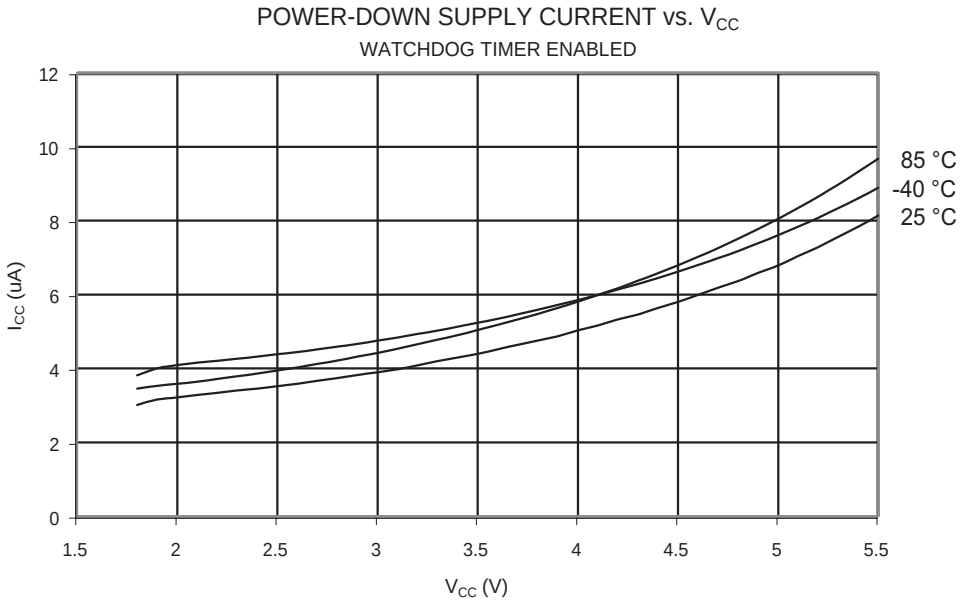
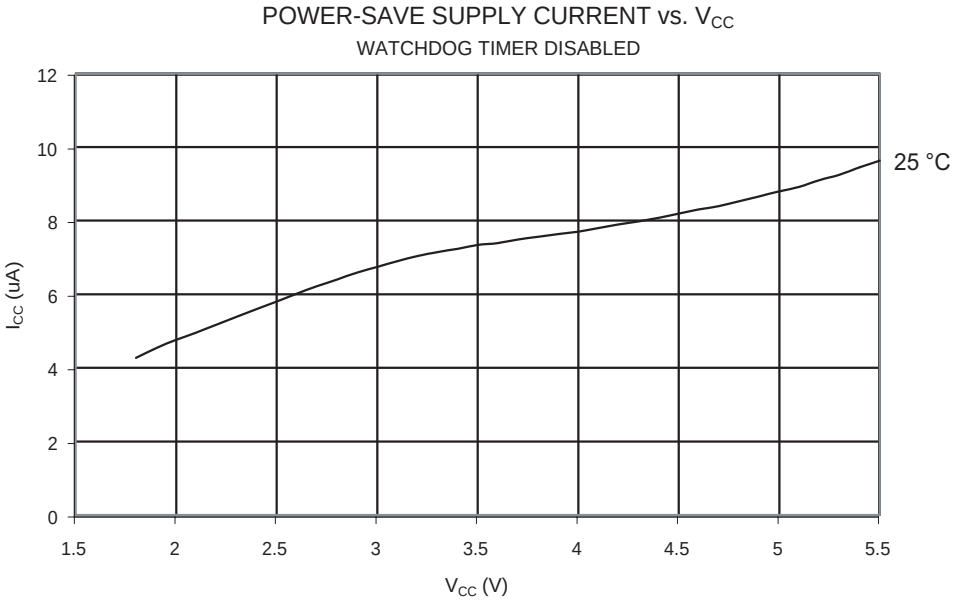


Figure 147. 掉电模式电流和 V_{CC} 的关系 (看门狗定时器使能)



省电模式电流

Figure 148. 省电模式电流和 V_{CC} 的关系 (看门狗定时器禁用)



Standby 模式电流

Figure 149. Standby 模式电流和 V_{CC} 的关系

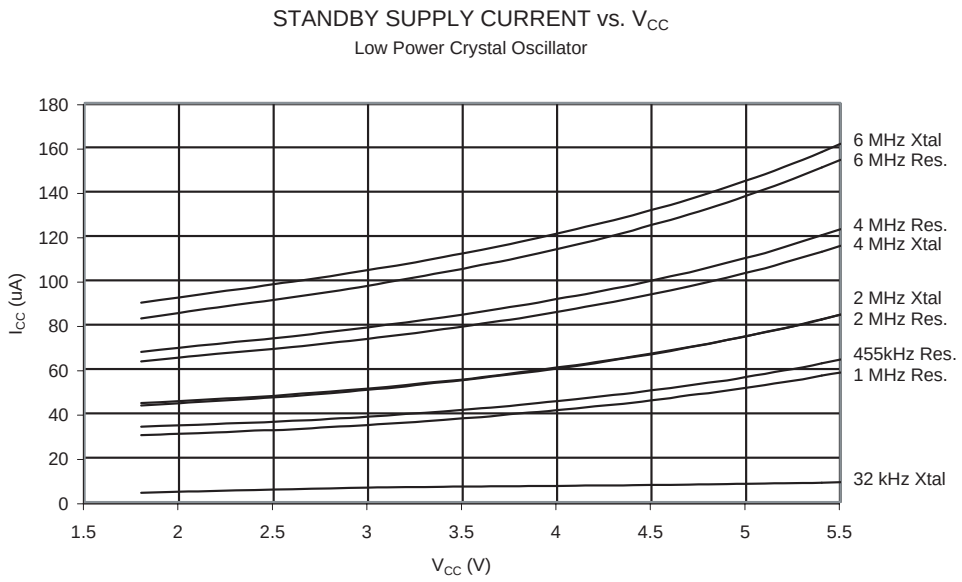
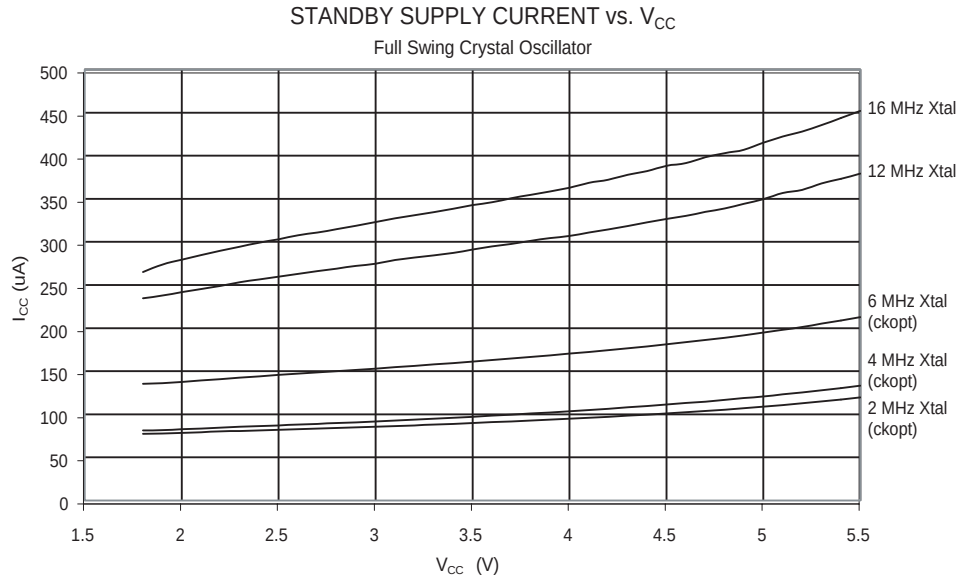


Figure 150. Standby 模式电流和 V_{CC} 的关系



引脚上拉

Figure 151. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 5V)

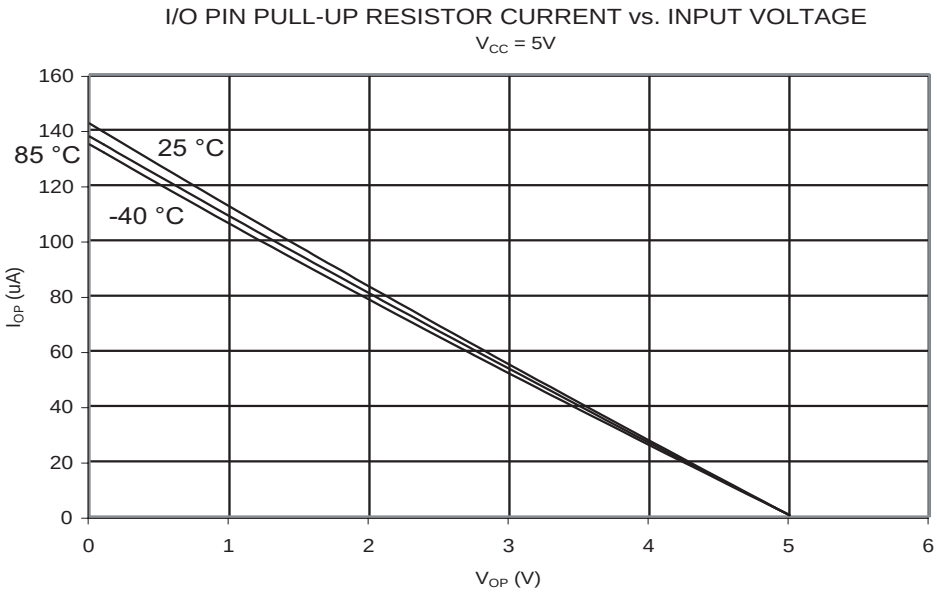


Figure 152. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 2.7V)

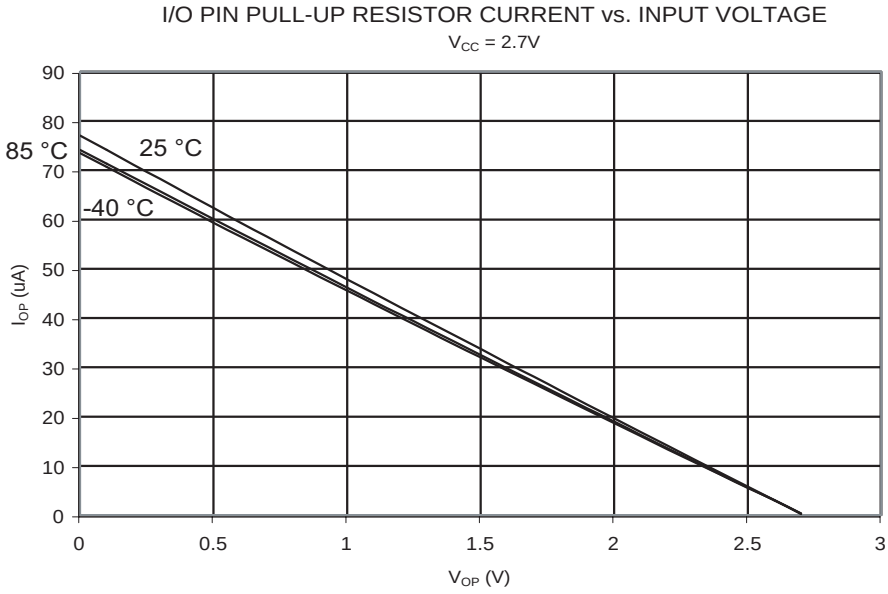


Figure 153. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 5V)

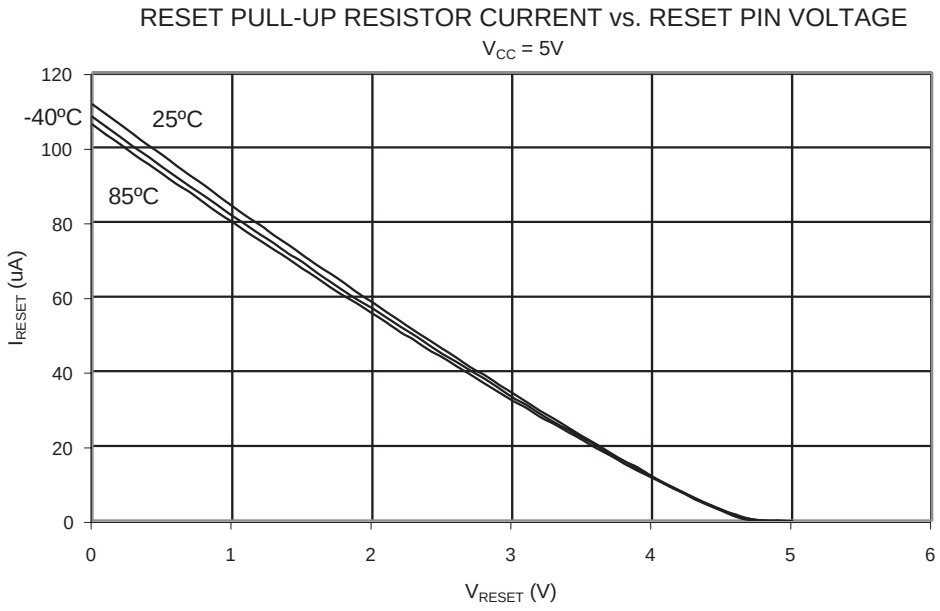
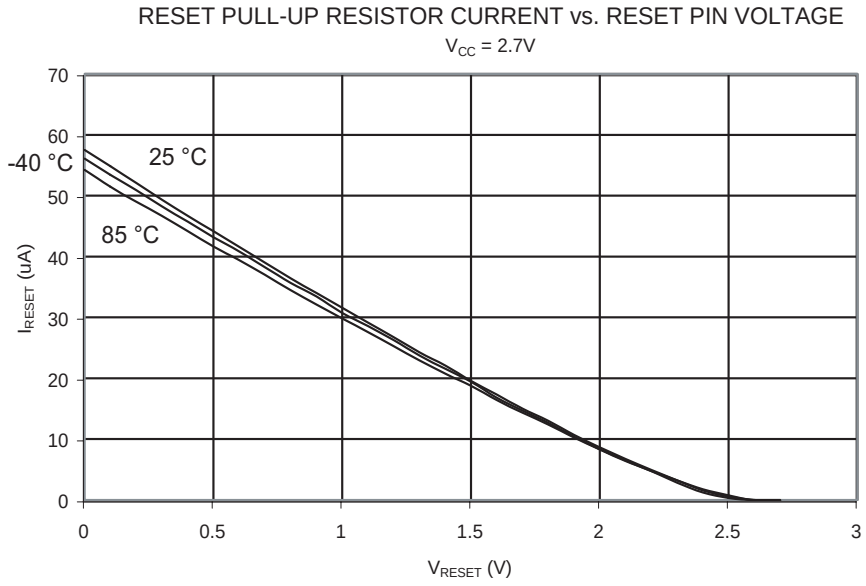


Figure 154. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 2.7V)



驱动能力

Figure 155. I/O 引脚源电流和输出电压的关系：端口 B、PC5..0、端口 D (V_{CC}= 5V)

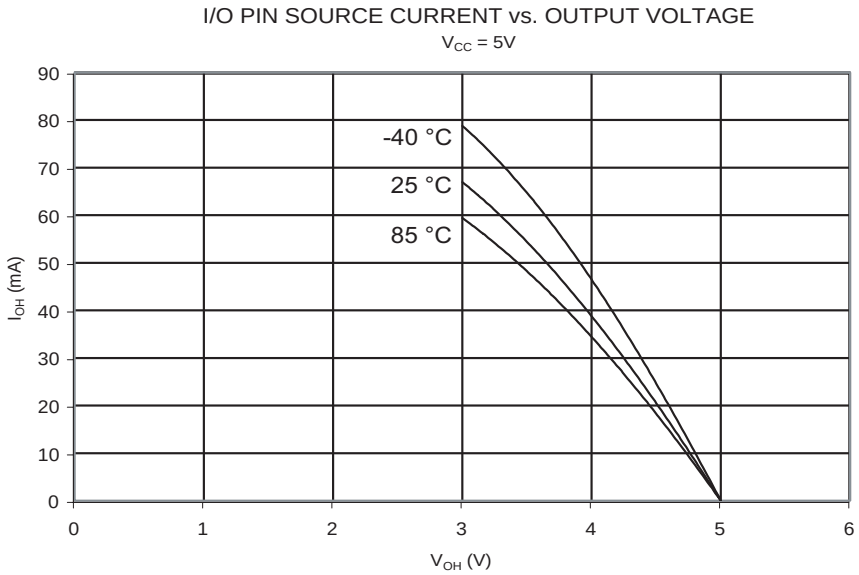


Figure 156. I/O 引脚源电流和输出电压的关系：端口 B、PC5..0、端口 D (V_{CC} = 2.7V)

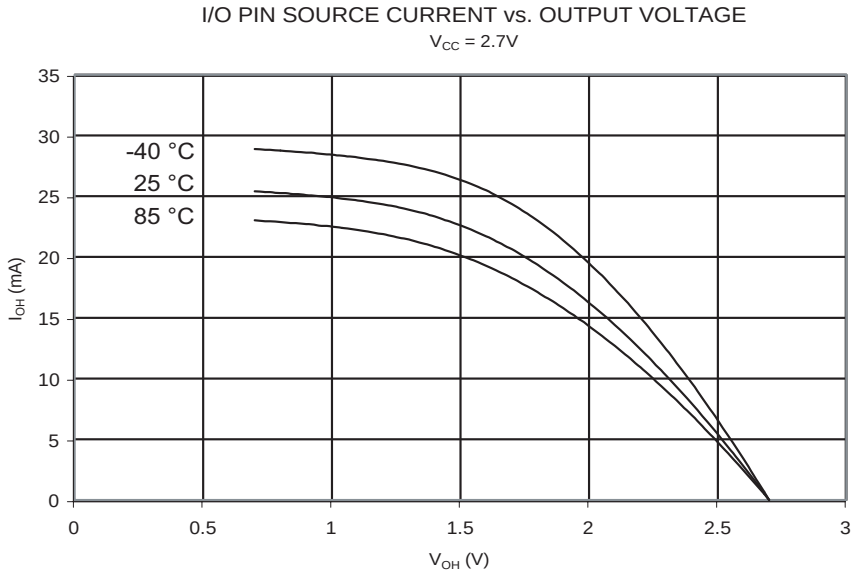


Figure 157. 引脚源电流和输出电压的关系：端口 B、PC5..0、端口 D (V_{CC} = 1.8V)

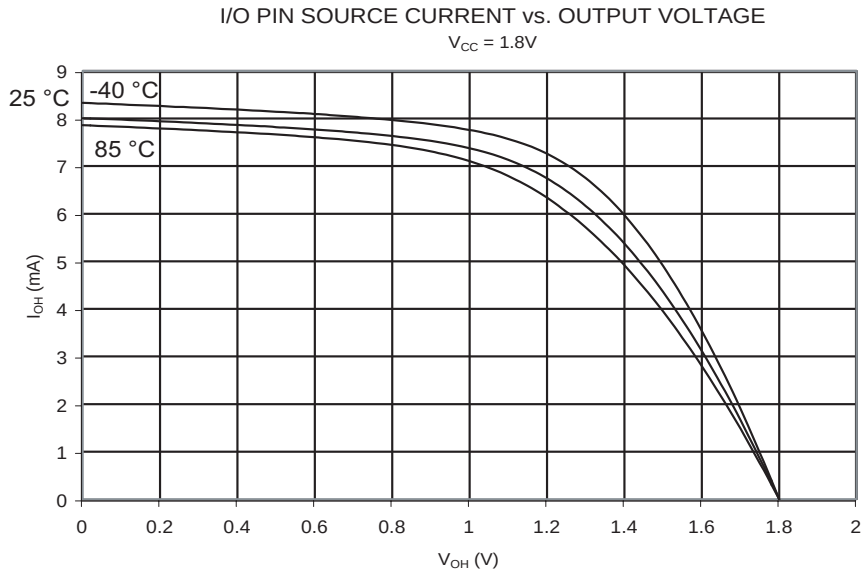


Figure 158. I/O 引脚吸收电流和输出电压的关系：端口 B、PC5..0、端口 D (V_{CC} = 5V)

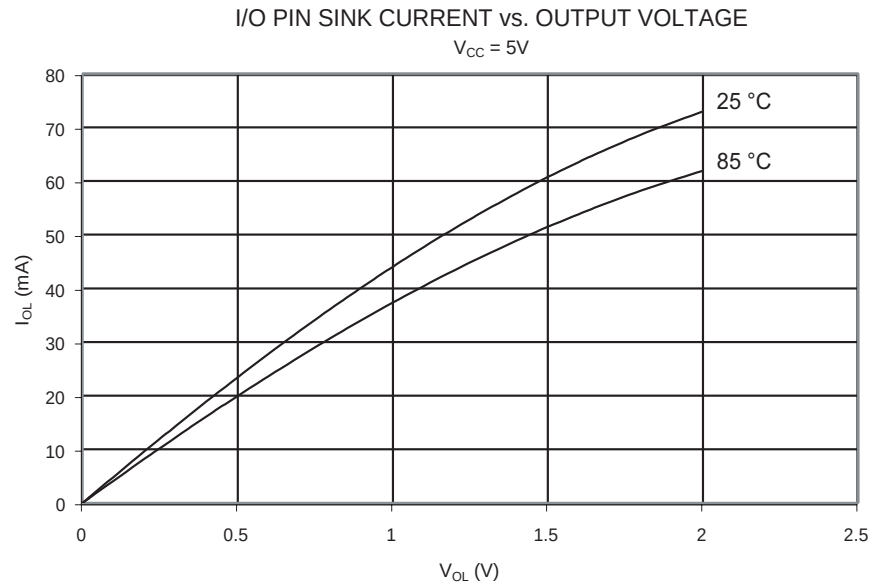


Figure 159. I/O 引脚吸收电流和输出电压的关系：端口 B、PC5..0、端口 D (V_{CC} = 2.7V)

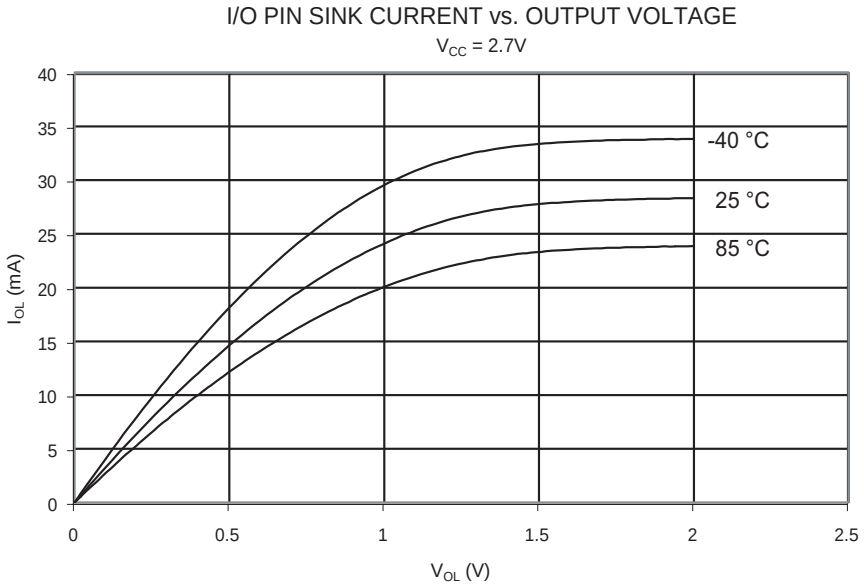
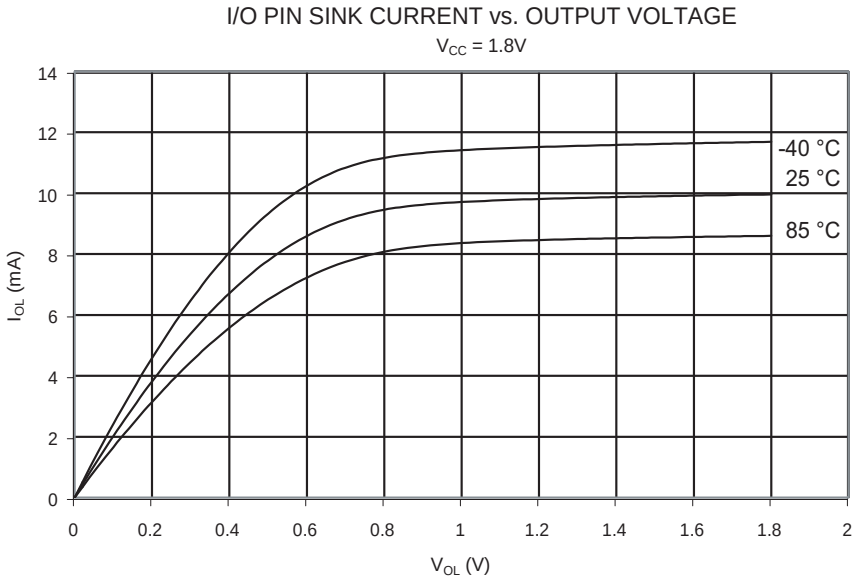


Figure 160. I/O 引脚吸收电流和输出电压的关系：端口 B、PC5..0、端口 D (V_{CC} = 1.8V)



引脚门限及滞后

Figure 161. I/O 引脚输入门限电压和 V_{CC} 的关系 (V_{IH}, I/O 引脚读出值为 “1”)

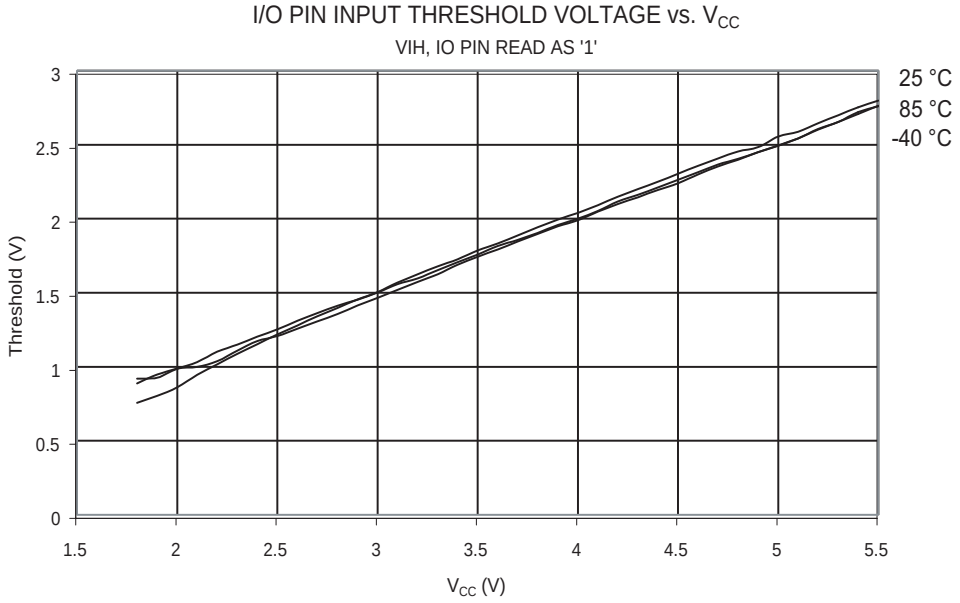


Figure 162. I/O 引脚输入门限电压和 V_{CC} 的关系 (V_{IL}, I/O 引脚读出值为 “0”)

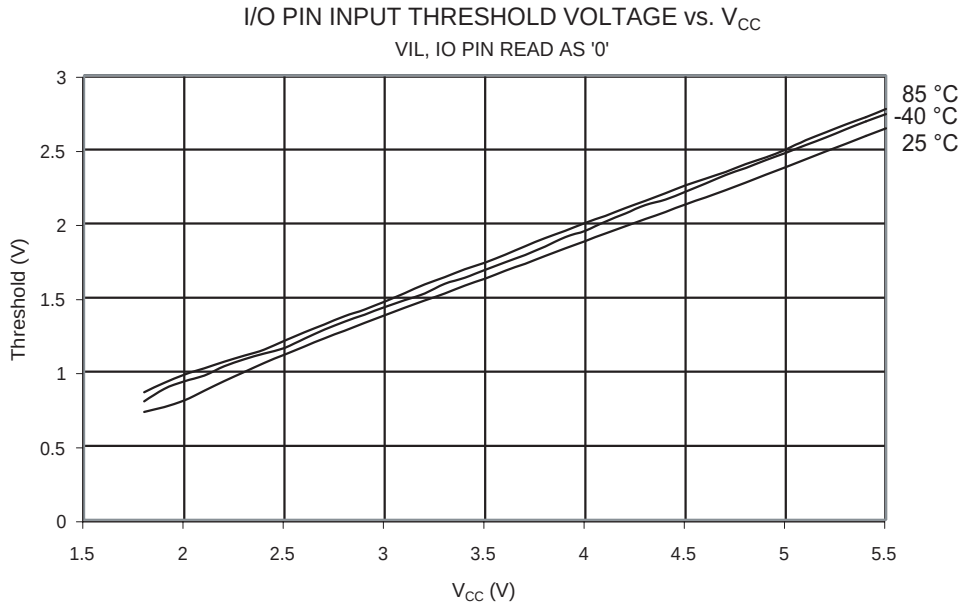


Figure 163. Reset 输入门限电压和 V_{CC} 的关系 (V_{IH},Reset 引脚读出值为 “1”)

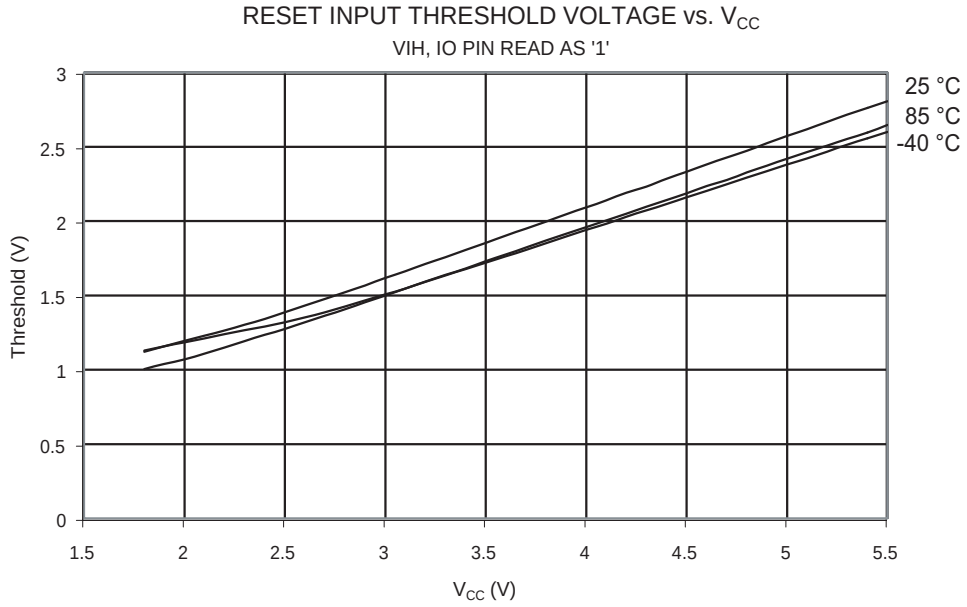


Figure 164. Reset 输入门限电压和 V_{CC} 的关系 (V_{IL},Reset 引脚读出值为 “0”)

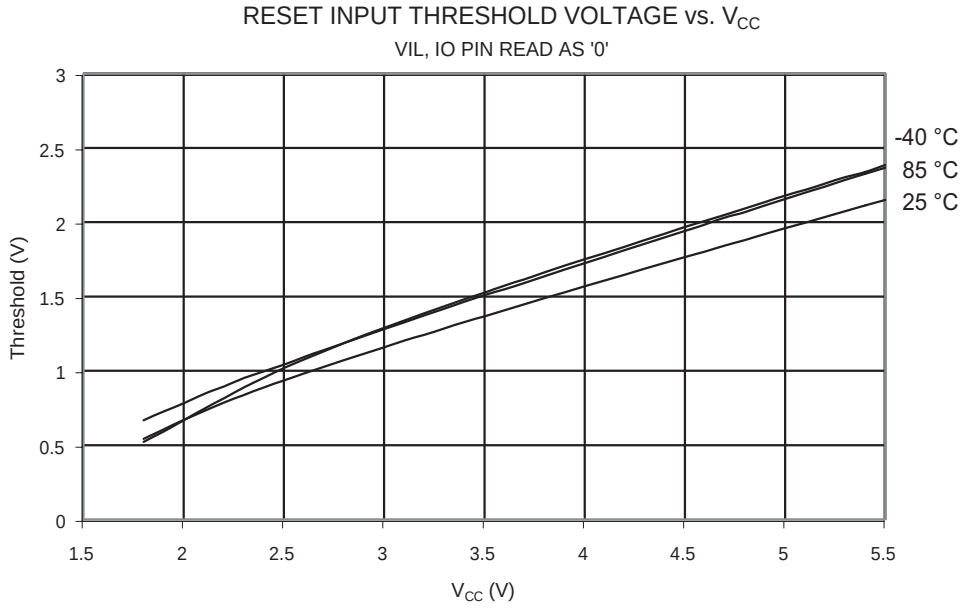
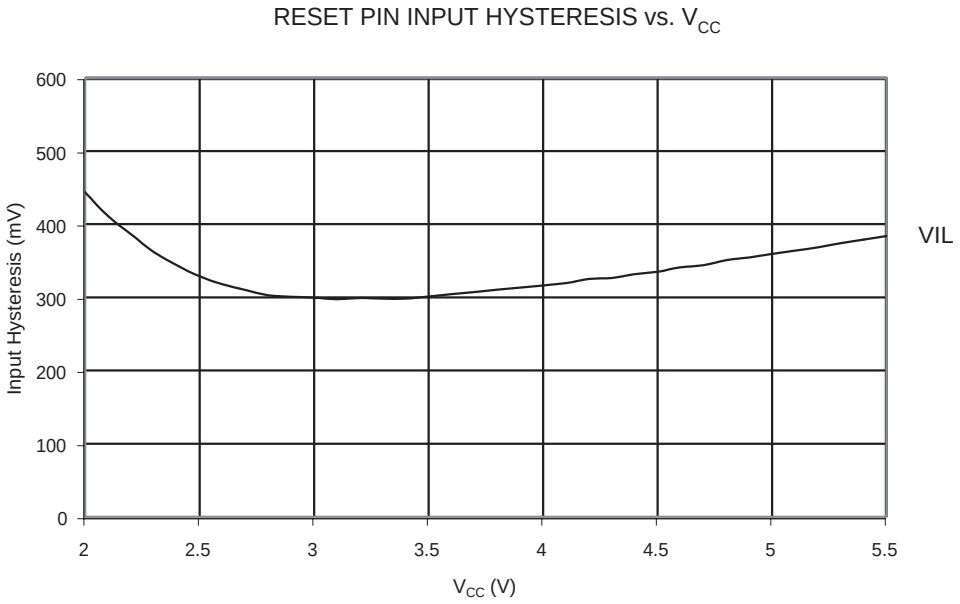


Figure 165. Reset 输入迟滞和 V_{CC} 的关系



BOD 门限值与模拟比较器偏移量

Figure 166. BOD 门限值和温度的关系 (BOD 电平为 4.0V)

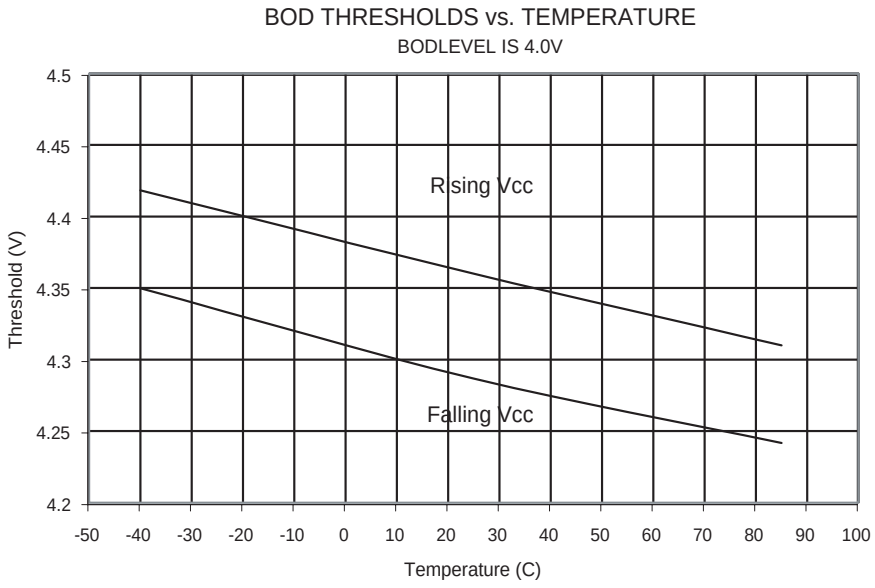


Figure 167. BOD 门限值和温度的关系 (BOD 电平为 2.7V)

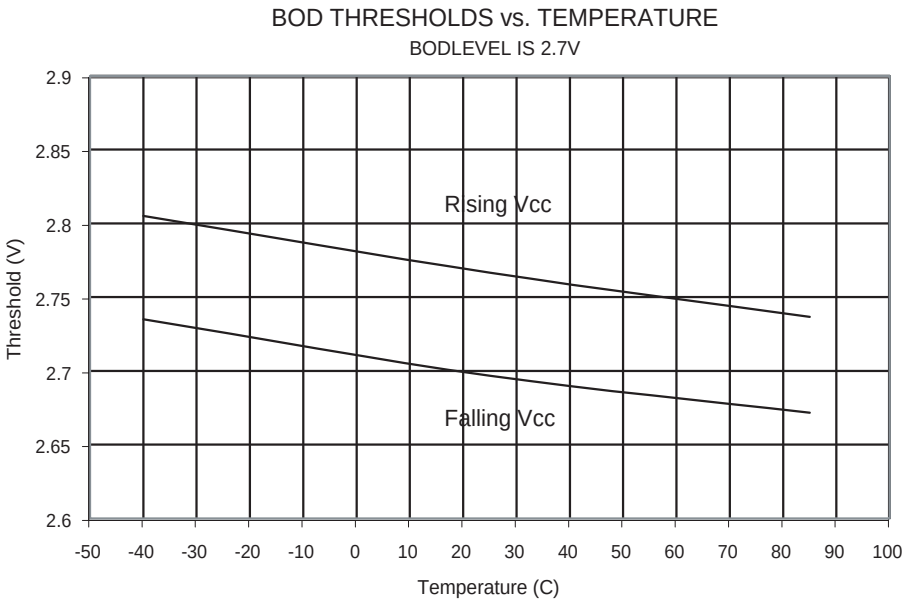


Figure 168. BOD 门限值和温度的关系 (BOD 电平为 1.8V)

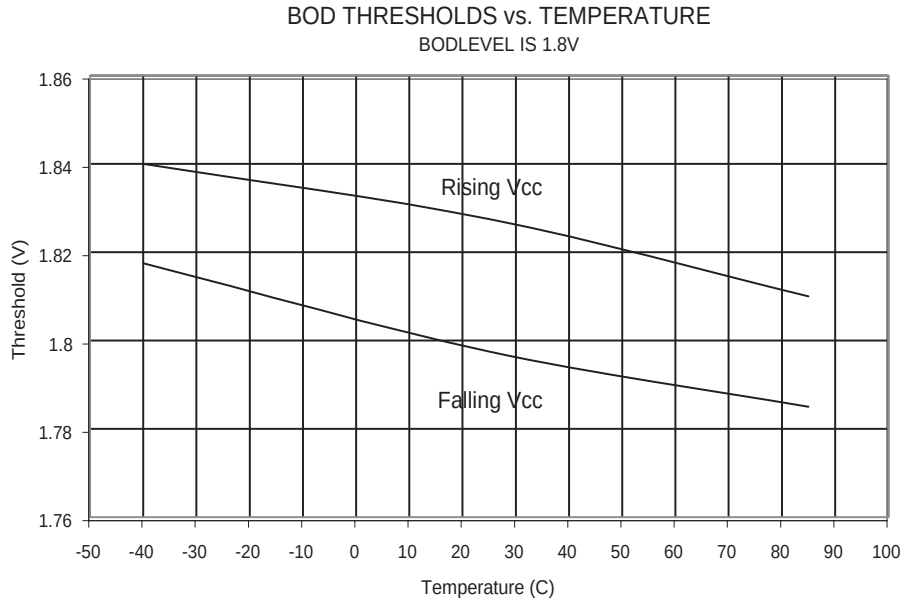


Figure 169. 能隙电压和 V_{CC} 的关系

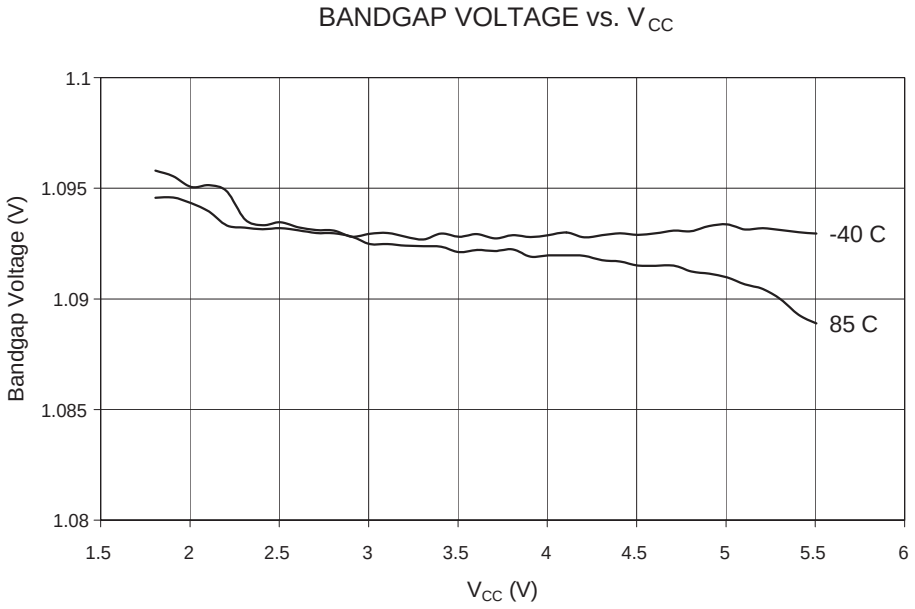


Figure 170. 模拟比较器偏置电压和共模电压的关系 (V_{CC} = 5V)

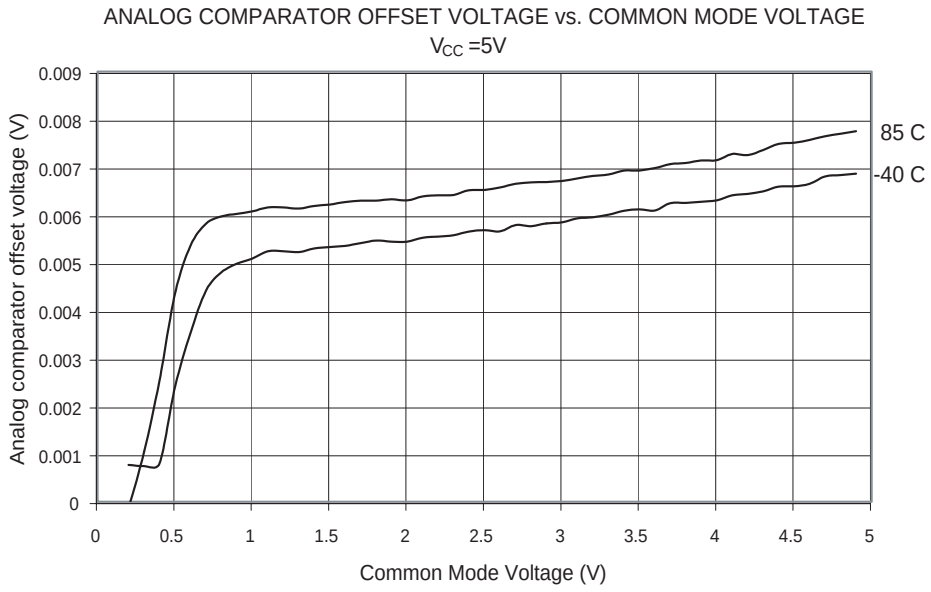
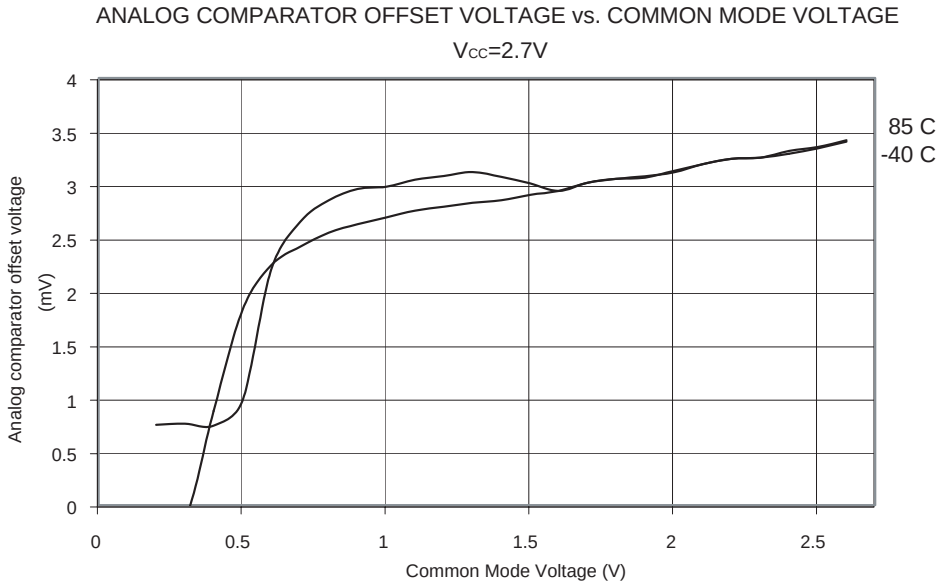


Figure 171. 模拟比较器偏置电压和共模电压的关系 (V_{CC} = 2.7V)



外设电流

Figure 172. BOD 电流和 V_{CC} 的关系

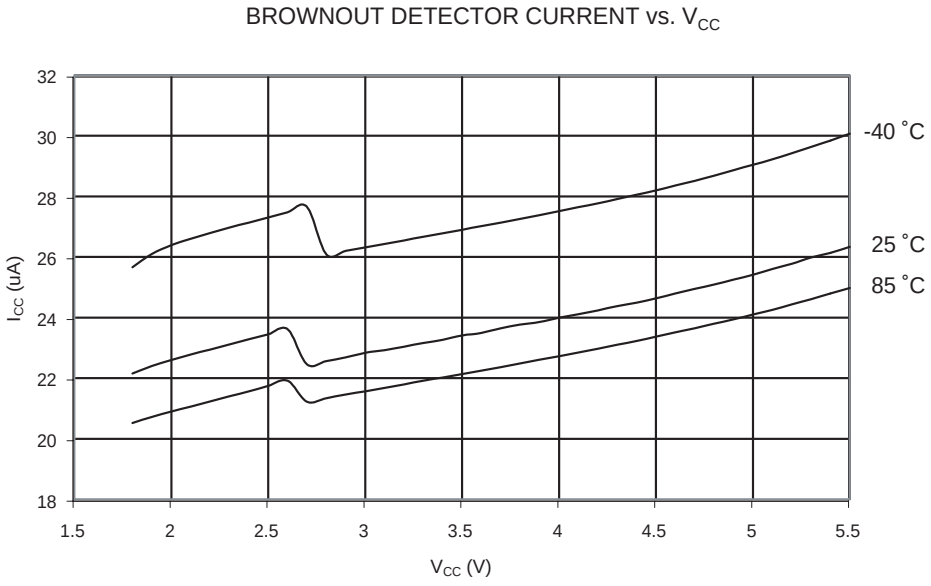


Figure 173. ADC 电流和 V_{CC} 的关系 (AREF = AVCC)

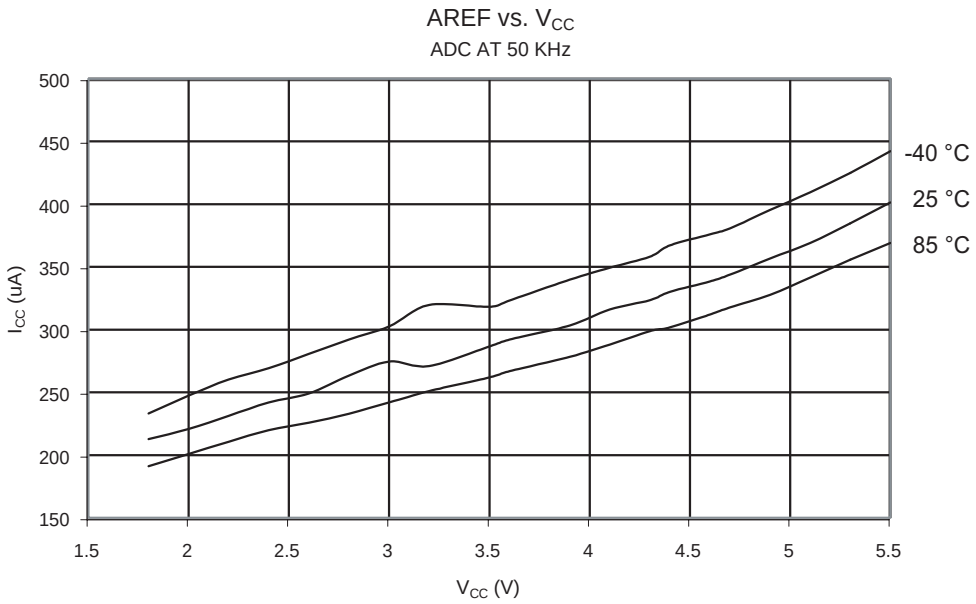


Figure 174. AREF 外部参考电流和 V_{CC} 的关系

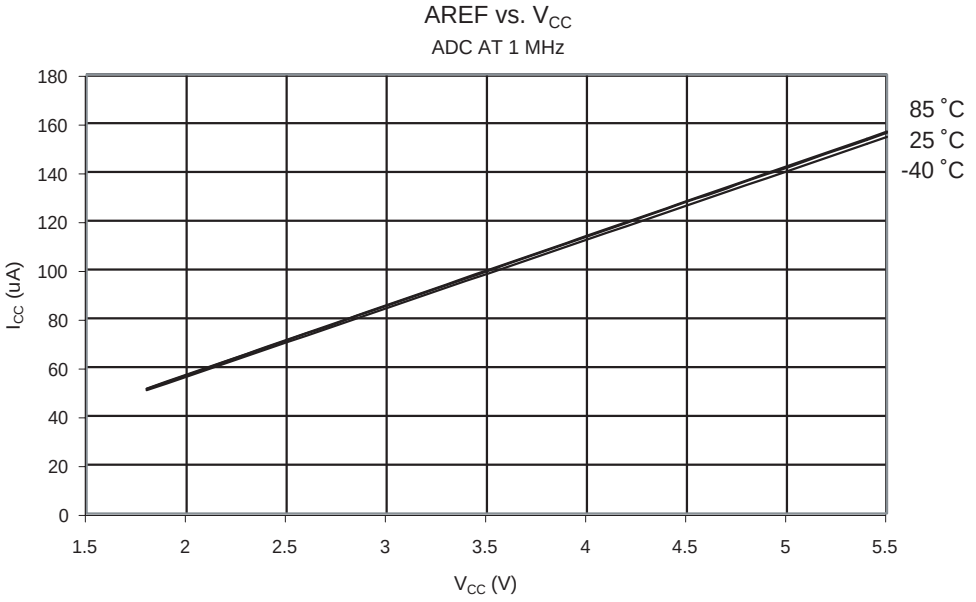


Figure 175. 模拟比较器电流和 V_{CC} 的关系

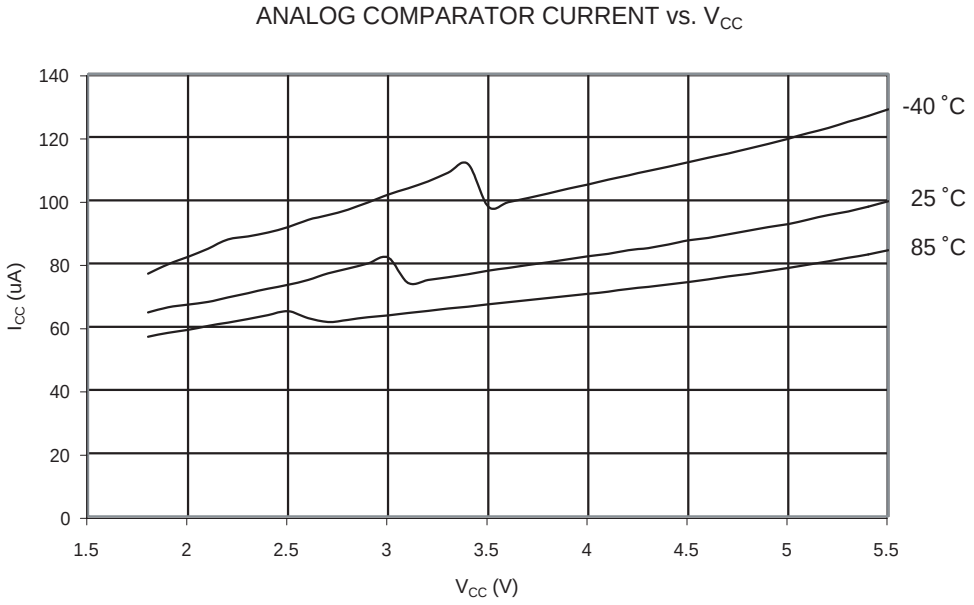


Figure 176. 模拟比较器电流和 V_{CC} 的关系

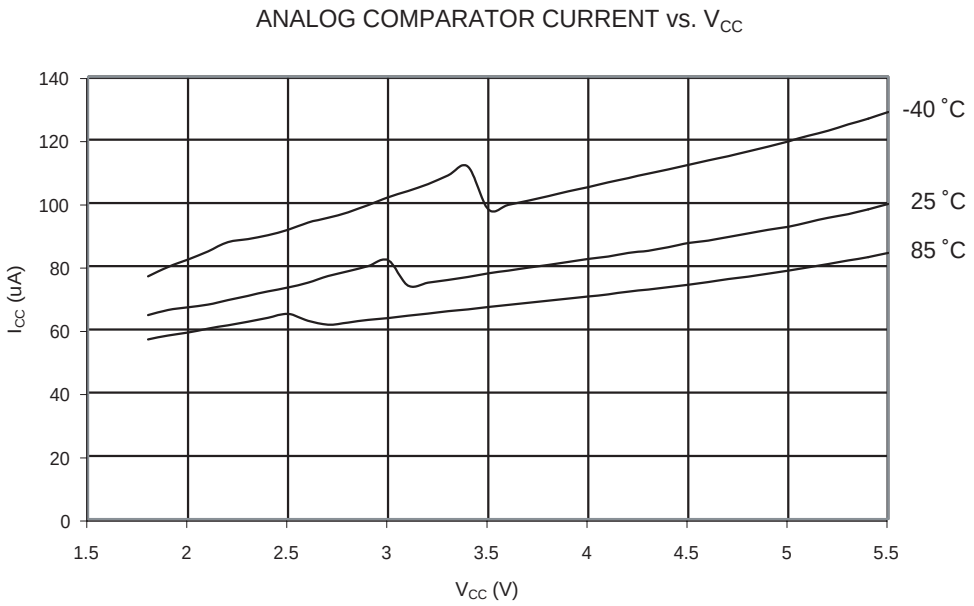
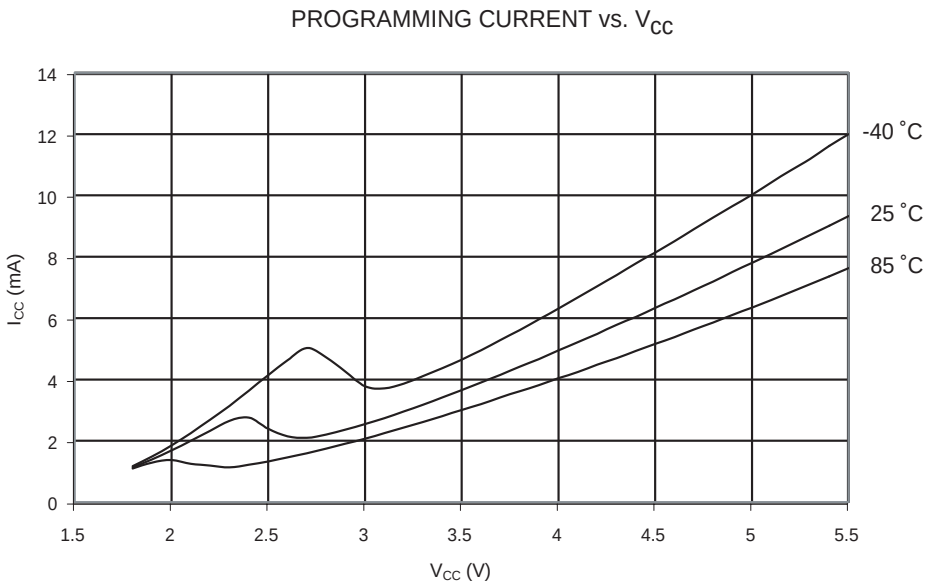


Figure 177. Programming Current vs. V_{CC}



复位电流消耗及复位脉宽

Figure 178. 复位电流和 V_{CC} 的关系 (0.1 - 1.0 MHz，包括流经复位上拉电阻的电流)

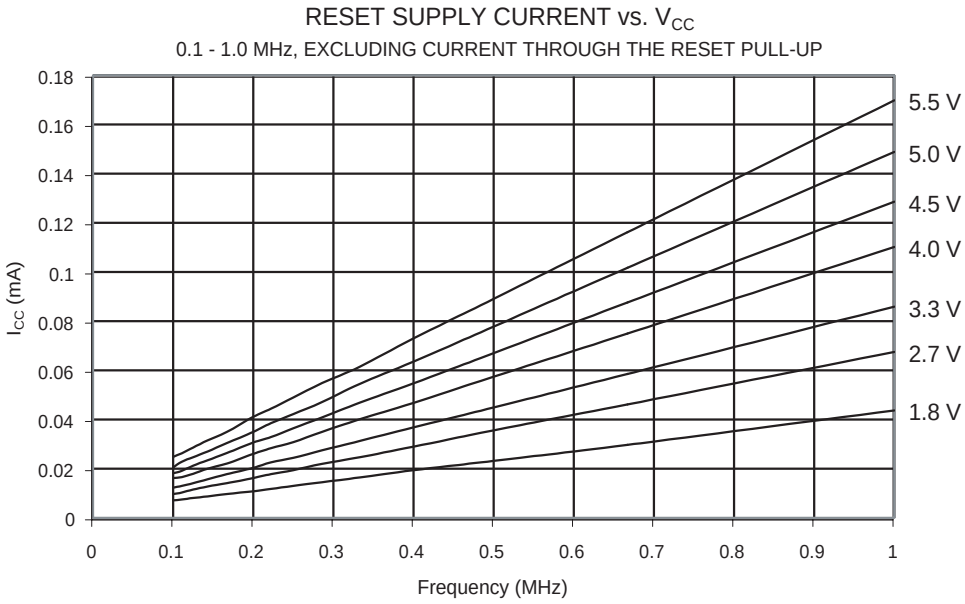


Figure 179. 复位电流和 V_{CC} 的关系 (1 - 20 MHz，包括流经复位上拉电阻的电流)

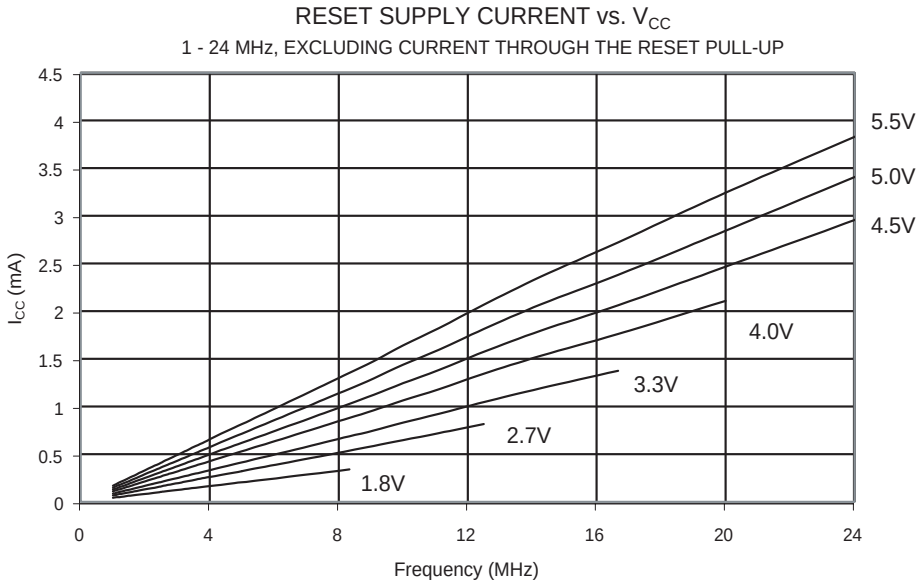
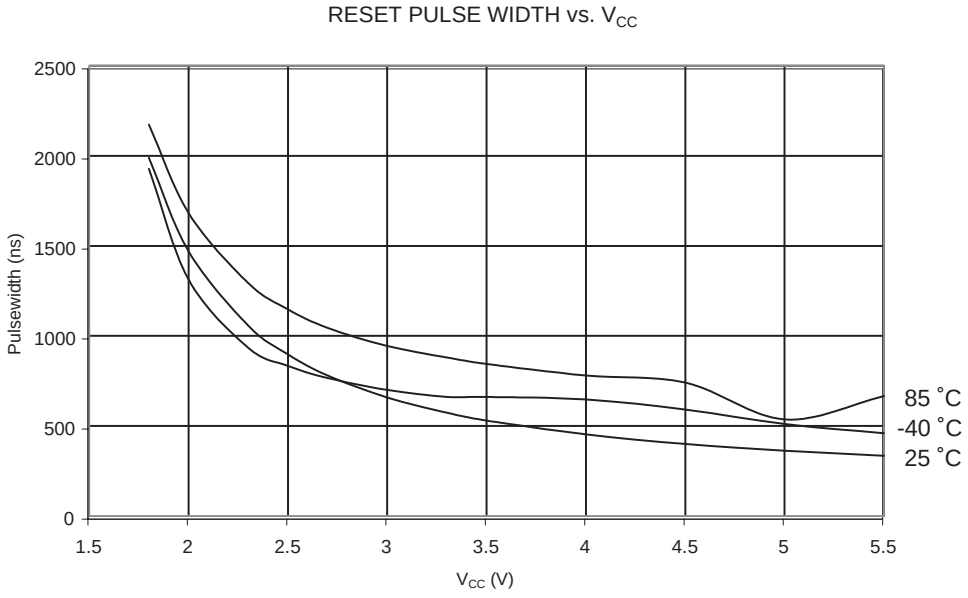


Figure 180. 复位脉宽和 V_{CC} 的关系



寄存器概述

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
(0xFF)	保留	–	–	–	–	–	–	–	–	
(0xFE)	保留	–	–	–	–	–	–	–	–	
(0xFD)	保留	–	–	–	–	–	–	–	–	
(0xFC)	保留	–	–	–	–	–	–	–	–	
(0xFB)	保留	–	–	–	–	–	–	–	–	
(0xFA)	保留	–	–	–	–	–	–	–	–	
(0xF9)	保留	–	–	–	–	–	–	–	–	
(0xF8)	保留	–	–	–	–	–	–	–	–	
(0xF7)	保留	–	–	–	–	–	–	–	–	
(0xF6)	保留	–	–	–	–	–	–	–	–	
(0xF5)	保留	–	–	–	–	–	–	–	–	
(0xF4)	保留	–	–	–	–	–	–	–	–	
(0xF3)	保留	–	–	–	–	–	–	–	–	
(0xF2)	保留	–	–	–	–	–	–	–	–	
(0xF1)	保留	–	–	–	–	–	–	–	–	
(0xF0)	保留	–	–	–	–	–	–	–	–	
(0xEF)	保留	–	–	–	–	–	–	–	–	
(0xEE)	保留	–	–	–	–	–	–	–	–	
(0xED)	保留	–	–	–	–	–	–	–	–	
(0xEC)	保留	–	–	–	–	–	–	–	–	
(0xEB)	保留	–	–	–	–	–	–	–	–	
(0xEA)	保留	–	–	–	–	–	–	–	–	
(0xE9)	保留	–	–	–	–	–	–	–	–	
(0xE8)	保留	–	–	–	–	–	–	–	–	
(0xE7)	保留	–	–	–	–	–	–	–	–	
(0xE6)	保留	–	–	–	–	–	–	–	–	
(0xE5)	保留	–	–	–	–	–	–	–	–	
(0xE4)	保留	–	–	–	–	–	–	–	–	
(0xE3)	保留	–	–	–	–	–	–	–	–	
(0xE2)	保留	–	–	–	–	–	–	–	–	
(0xE1)	保留	–	–	–	–	–	–	–	–	
(0xE0)	保留	–	–	–	–	–	–	–	–	
(0xDF)	保留	–	–	–	–	–	–	–	–	
(0xDE)	保留	–	–	–	–	–	–	–	–	
(0xDD)	保留	–	–	–	–	–	–	–	–	
(0xDC)	保留	–	–	–	–	–	–	–	–	
(0xDB)	保留	–	–	–	–	–	–	–	–	
(0xDA)	保留	–	–	–	–	–	–	–	–	
(0xD9)	保留	–	–	–	–	–	–	–	–	
(0xD8)	保留	–	–	–	–	–	–	–	–	
(0xD7)	保留	–	–	–	–	–	–	–	–	
(0xD6)	保留	–	–	–	–	–	–	–	–	
(0xD5)	保留	–	–	–	–	–	–	–	–	
(0xD4)	保留	–	–	–	–	–	–	–	–	
(0xD3)	保留	–	–	–	–	–	–	–	–	
(0xD2)	保留	–	–	–	–	–	–	–	–	
(0xD1)	保留	–	–	–	–	–	–	–	–	
(0xD0)	保留	–	–	–	–	–	–	–	–	
(0xCF)	保留	–	–	–	–	–	–	–	–	
(0xCE)	保留	–	–	–	–	–	–	–	–	
(0xCD)	保留	–	–	–	–	–	–	–	–	
(0xCC)	保留	–	–	–	–	–	–	–	–	
(0xCB)	保留	–	–	–	–	–	–	–	–	
(0xCA)	保留	–	–	–	–	–	–	–	–	
(0xC9)	保留	–	–	–	–	–	–	–	–	
(0xC8)	保留	–	–	–	–	–	–	–	–	
(0xC7)	保留	–	–	–	–	–	–	–	–	
(0xC6)	UDR0	USART I/O 数据寄存器 r								168
(0xC5)	UBRR0H					USART 波特率寄存器高位				172
(0xC4)	UBRR0L	USART 波特率寄存器低位								172
(0xC3)	保留	–	–	–	–	–	–	–	–	
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 /UDORD0	UCSZ00 / UCPHA0	UCPOL0	171/182
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	170
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	169

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
(0xBF)	保留	–	–	–	–	–	–	–	–	
(0xBE)	保留	–	–	–	–	–	–	–	–	
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–	195
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	192
(0xBB)	TWDR	两线串行接口数据寄存器								194
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	194
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	194
(0xB8)	TWBR	两线串行接口位率寄存器								192
(0xB7)	保留	–		–	–	–	–	–	–	
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB	139
(0xB5)	保留	–	–	–	–	–	–	–	–	
(0xB4)	OCR2B	定时器 / 计数器 2 输出比较寄存器 B								138
(0xB3)	OCR2A	定时器 / 计数器 2 输出比较寄存器 r A								138
(0xB2)	TCNT2	定时器 / 计数器 2(8 位)								138
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	137
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	134
(0xAF)	保留	–	–	–	–	–	–	–	–	
(0xAE)	保留	–	–	–	–	–	–	–	–	
(0xAD)	保留	–	–	–	–	–	–	–	–	
(0xAC)	保留	–	–	–	–	–	–	–	–	
(0xAB)	保留	–	–	–	–	–	–	–	–	
(0xAA)	保留	–	–	–	–	–	–	–	–	
(0xA9)	保留	–	–	–	–	–	–	–	–	
(0xA8)	保留	–	–	–	–	–	–	–	–	
(0xA7)	保留	–	–	–	–	–	–	–	–	
(0xA6)	保留	–	–	–	–	–	–	–	–	
(0xA5)	保留	–	–	–	–	–	–	–	–	
(0xA4)	保留	–	–	–	–	–	–	–	–	
(0xA3)	保留	–	–	–	–	–	–	–	–	
(0xA2)	保留	–	–	–	–	–	–	–	–	
(0xA1)	保留	–	–	–	–	–	–	–	–	
(0xA0)	保留	–	–	–	–	–	–	–	–	
(0x9F)	保留	–	–	–	–	–	–	–	–	
(0x9E)	保留	–	–	–	–	–	–	–	–	
(0x9D)	保留	–	–	–	–	–	–	–	–	
(0x9C)	保留	–	–	–	–	–	–	–	–	
(0x9B)	保留	–	–	–	–	–	–	–	–	
(0x9A)	保留	–	–	–	–	–	–	–	–	
(0x99)	保留	–	–	–	–	–	–	–	–	
(0x98)	保留	–	–	–	–	–	–	–	–	
(0x97)	保留	–	–	–	–	–	–	–	–	
(0x96)	保留	–	–	–	–	–	–	–	–	
(0x95)	保留	–	–	–	–	–	–	–	–	
(0x94)	保留	–	–	–	–	–	–	–	–	
(0x93)	保留	–	–	–	–	–	–	–	–	
(0x92)	保留	–	–	–	–	–	–	–	–	
(0x91)	保留	–	–	–	–	–	–	–	–	
(0x90)	保留	–	–	–	–	–	–	–	–	
(0x8F)	保留	–	–	–	–	–	–	–	–	
(0x8E)	保留	–	–	–	–	–	–	–	–	
(0x8D)	保留	–	–	–	–	–	–	–	–	
(0x8C)	保留	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	定时器 / 计数器 1 输出比较寄存器 B 高字节								121
(0x8A)	OCR1BL	定时器 / 计数器 1 输出比较寄存器 B 低字节								121
(0x89)	OCR1AH	定时器 / 计数器 1 输出比较寄存器 A 高字节								121
(0x88)	OCR1AL	定时器 / 计数器 1 输出比较寄存器 A 低字节								121
(0x87)	ICR1H	定时器 / 计数器 1 输入捕获寄存器高字节								122
(0x86)	ICR1L	定时器 / 计数器 1 输入捕获寄存器低字节								122
(0x85)	TCNT1H	定时器 / 计数器 1 计数器寄存器高字节								120
(0x84)	TCNT1L	定时器 / 计数器 1 计数器寄存器低字节								120
(0x83)	保留	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	120
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	119
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	117
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D	216
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	231

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
(0x7D)	保留	–	–	–	–	–	–	–	–	
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	227
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	231
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	228
(0x79)	ADCH	ADC 数据寄存器高字节								229
(0x78)	ADCL	ADC 数据寄存器低字节								229
(0x77)	保留	–	–	–	–	–	–	–	–	
(0x76)	保留	–	–	–	–	–	–	–	–	
(0x75)	保留	–	–	–	–	–	–	–	–	
(0x74)	保留	–	–	–	–	–	–	–	–	
(0x73)	保留	–	–	–	–	–	–	–	–	
(0x72)	保留	–	–	–	–	–	–	–	–	
(0x71)	保留	–	–	–	–	–	–	–	–	
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	140
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	122
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	93
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	79
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	79
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	79
(0x6A)	保留	–	–	–	–	–	–	–	–	
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	76
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0	
(0x67)	保留	–	–	–	–	–	–	–	–	
(0x66)	OSCCAL	振荡器寄存器 r								29
(0x65)	保留	–	–	–	–	–	–	–	–	
(0x64)	PRR	PRTW1	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	
(0x63)	保留	–	–	–	–	–	–	–	–	
(0x62)	保留	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	31
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	44
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	8
0x3E (0x5E)	SPH	–	–	–	–	–	(SP10) [5.]	SP9	SP8	10
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
0x3C (0x5C)	保留	–	–	–	–	–	–	–	–	
0x3B (0x5B)	保留	–	–	–	–	–	–	–	–	
0x3A (0x5A)	保留	–	–	–	–	–	–	–	–	
0x39 (0x59)	保留	–	–	–	–	–	–	–	–	
0x38 (0x58)	保留	–	–	–	–	–	–	–	–	
0x37 (0x57)	SPMCSR	SPMIE	(RWWSB)[5.]	–	(RWWSRE)[5.]	BLBSET	PGWRT	PGERS	SPMEN	244
0x36 (0x56)	保留	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	–	–	–	PUD	–	–	IVSEL	IVCE	
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE	33
0x32 (0x52)	保留	–	–	–	–	–	–	–	–	
0x31 (0x51)	MONDR	监控数据寄存器								
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	214
0x2F (0x4F)	保留	–	–	–	–	–	–	–	–	
0x2E (0x4E)	SPDR	SPI 数据寄存器								150
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	150
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	148
0x2B (0x4B)	GPIOR2	通用 I/O 寄存器 2								22
0x2A (0x4A)	GPIOR1	通用 I/O 寄存器 1								22
0x29 (0x49)	保留	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	定时器 / 计数器 0 输出比较寄存器 B								
0x27 (0x47)	OCR0A	定时器 / 计数器 0 输出比较寄存器 A								
0x26 (0x46)	TCNT0	定时器 / 计数器 0(8 位)								
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC	96/142
0x22 (0x42)	EEARH	(EEPROM 地址寄存器高字节) [5.]								17
0x21 (0x41)	EEARL	EEPROM 地址寄存器低字节								17
0x20 (0x40)	EEDR	EEPROM 数据寄存器								17
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	17
0x1E (0x3E)	GPIOR0	通用 I/O 寄存器 0								22
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	77
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	77

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0	
0x1A (0x3A)	保留	–	–	–	–	–	–	–	–	
0x19 (0x39)	保留	–	–	–	–	–	–	–	–	
0x18 (0x38)	保留	–	–	–	–	–	–	–	–	
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2	141
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	123
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	保留	–	–	–	–	–	–	–	–	
0x13 (0x33)	保留	–	–	–	–	–	–	–	–	
0x12 (0x32)	保留	–	–	–	–	–	–	–	–	
0x11 (0x31)	保留	–	–	–	–	–	–	–	–	
0x10 (0x30)	保留	–	–	–	–	–	–	–	–	
0x0F (0x2F)	保留	–	–	–	–	–	–	–	–	
0x0E (0x2E)	保留	–	–	–	–	–	–	–	–	
0x0D (0x2D)	保留	–	–	–	–	–	–	–	–	
0x0C (0x2C)	保留	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	75
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	75
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	75
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	75
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	75
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	75
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	75
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	75
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	75
0x02 (0x22)	保留	–	–	–	–	–	–	–	–	
0x01 (0x21)	保留	–	–	–	–	–	–	–	–	
0x0 (0x20)	保留	–	–	–	–	–	–	–	–	

Note:
1. 为了和未来的设备兼容，如果需要保留位应该被写 "0”。保留的 I/O 地址不可以写入。
2. 通过 SBI 和 CBI 指令可直接对地址从 0x00 - 0x1F 的 I/O 寄存器进行位寻址。在这些寄存器中 ,单个位的值可以通过 SBIS 和 SBIC 指 令查询。
3. 一些状态标志可以通过写入逻辑"1"来清空。需要注意的是 ,不同于大多数其他的 AVR ,CBI 和 SBI 指令只对一些特殊位有效，因 此可以对那些包含标志位的寄存器进行操作。CBI 和 SBI 指令可使用的范围只能是地址从 0x00 - 0x1F 的寄存器。
4. 当使用特殊的I/O操作命令IN和OUT时 ,从0x00 - 0x3F的I/O地址会被用到。使用LD 和ST命令可以像操作普通数据空间一样对 I/O 寄存器进行寻址 ,这时要增加地址 0x20 。ATmega48/88/168 是一款拥有更多外围设备单元的复杂的微处理器 ,可以使用 64 位的 IN 和 OUT 指令。SRAM 中的 0x60 - 0xFF 地址作为扩展 I/O 用 ，只有 ST/STS/STD 和 LD/LDS/LDD 指令可用。
5. 只对 ATmega88/168 有效

指令集概述

指令	操作数	说明	操作	标志	# 时钟数
		算术和逻辑指令			
ADD	Rd, Rr	无进位加法	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	带进位加法	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	立即数与字相加	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	无进位减法	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	减立即数	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	带进位减法	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	带进位减立即数	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	从字中减立即数	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	逻辑与	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	与立即数的逻辑与操作	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	逻辑或	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	与立即数的逻辑或操作	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	异或	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	1 的补码	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	2 的补码	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	设置寄存器的位	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	寄存器位清零	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	加一操作	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	减一操作	Rd ← Rd − 1	Z,N,V	1
TST	Rd	测试是否为零或负	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	寄存器清零	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	寄存器置位	Rd ← 0xFF	None	1
MUL	Rd, Rr	无符号数乘法	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	有符号数乘法	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	有符号数与无符号数乘法	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	无符号小数乘法	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	有符号小数乘法	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	有符号小数与无符号小数乘法	R1:R0 ← (Rd x Rr) << 1	Z,C	2
		跳转指令			
RJMP	k	相对跳转	PC ← PC + k + 1	None	2
IJMP		间接跳转到 (Z)	PC ← Z	None	2
JMP[1]	k	跳转	PC ← k	None	3
RCALL	k	相对子程序调用	PC ← PC + k + 1	None	3
ICALL		间接调用 (Z)	PC ← Z	None	3
CALL[1]	k	调用子程序	PC ← k	None	4
RET		子程序返回	PC ← STACK	None	4
RETI		中断返回	PC ← STACK	I	4
CPSE	Rd,Rr	比较，相等则跳过下一条指令	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	比较	Rd − Rr	Z, N,V,C,H	1
CPC	Rd,Rr	带进位比较	Rd − Rr − C	Z, N,V,C,H	1
CPI	Rd,K	与立即数比较	Rd − K	Z, N,V,C,H	1
SBRC	Rr, b	寄存器位为 "0" 则跳过下一条指令	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	寄存器位为 "1" 则跳过下一条指令	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	I/O 寄存器位为 "0" 则跳过下一条指令	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	I/O 寄存器位为 "1" 则跳过下一条指令	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	状态寄存器位为 "1" 则跳过下一条指令	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	状态寄存器位为 "0" 则跳过下一条指令	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	相等则跳转	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	不相等则跳转	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	进位位为 "1" 则跳转	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	进位位为 "0" 则跳转	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	大于或等于则跳转	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	小于则跳转	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	负则跳转	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	正则跳转	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	有符号数大于或等于则跳转	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	有符号数负则跳转	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	半进位位为 "1" 则跳转	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	半进位位为 "0" 则跳转	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	T 为 "1" 则跳转	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	T 为 "0" 则跳转	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	溢出标志为 "1" 则跳转	if (V = 1) then PC ← PC + k + 1	None	1/2

指令	操作数	说明	操作	标志	# 时钟数
BRVC	k	溢出标志为 "0" 则跳转	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	中断使能则跳转	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	中断禁止则跳转	if (I = 0) then PC ← PC + k + 1	None	1/2
		位和位测试指令			
SBI	P,b	I/O 寄存器位置位	I/O(P,b) ← 1	None	2
CBI	P,b	I/O 寄存器位清零	I/O(P,b) ← 0	None	2
LSL	Rd	逻辑左移	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	逻辑右移	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	带进位循环左移	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	带进位循环右移	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	算术右移	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	高低半字节交换	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	标志置位	SREG(s) ← 1	SREG(s)	1
BCLR	s	标志清零	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	从寄存器将位赋给 T	T ← Rr(b)	T	1
BLD	Rd, b	将 T 赋给寄存器位	Rd(b) ← T	None	1
SEC		进位位置位	C ← 1	C	1
CLC		进位位清零	C ← 0	C	1
SEN		负标志位置位	N ← 1	N	1
CLN		负标志位清零	N ← 0	N	1
SEZ		零标志位置位	Z ← 1	Z	1
CLZ		零标志位清零	Z ← 0	Z	1
SEI		全局中断使能	I ← 1	I	1
CLI		全局中断禁用	I ← 0	I	1
SES		符号测试标志位置位	S ← 1	S	1
CLS		符号测试标志位清零	S ← 0	S	1
SEV		2 的补码溢出标志置位	V ← 1	V	1
CLV		2 的补码溢出标志清零	V ← 0	V	1
SET		SREG 的 T 置位	T ← 1	T	1
CLT		SREG 的 T 清零	T ← 0	T	1
SEH		SREG 的半进位标志置位	H ← 1	H	1
CLH		SREG 的半进位标志清零	H ← 0	H	1
		数据传送指令			
MOV	Rd, Rr	寄存器间复制	Rd ← Rr	None	1
MOVW	Rd, Rr	复制寄存器字	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	加载立即数	Rd ← K	None	1
LD	Rd, X	加载间接寻址数据	Rd ← (X)	None	2
LD	Rd, X+	加载间接寻址数据，然后地址加一	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	地址减一后加载间接寻址数据	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	加载间接寻址数据	Rd ← (Y)	None	2
LD	Rd, Y+	加载间接寻址数据，然后地址加一	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	地址减一后加载间接寻址数据	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	加载带偏移量的间接寻址数据	Rd ← (Y + q)	None	2
LD	Rd, Z	加载间接寻址数据	Rd ← (Z)	None	2
LD	Rd, Z+	加载间接寻址数据，然后地址加一	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	地址减一后加载间接寻址数据	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	加载带偏移量的间接寻址数据	Rd ← (Z + q)	None	2
LDS	Rd, k	从 SRAM 加载数据	Rd ← (k)	None	2
ST	X, Rr	以间接寻址方式存储数据	(X) ← Rr	None	2
ST	X+, Rr	以间接寻址方式存储数据，然后地址加一	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	地址减一后以间接寻址方式存储数据	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	加载间接寻址数据	(Y) ← Rr	None	2
ST	Y+, Rr	加载间接寻址数据，然后地址加一	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	地址减一后加载间接寻址数据	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	加载带偏移量的间接寻址数据	(Y + q) ← Rr	None	2
ST	Z, Rr	加载间接寻址数据	(Z) ← Rr	None	2
ST	Z+, Rr	加载间接寻址数据，然后地址加一	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	地址减一后加载间接寻址数据	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	加载带偏移量的间接寻址数据	(Z + q) ← Rr	None	2
STS	k, Rr	从 SRAM 加载数据	(k) ← Rr	None	2
LPM		加载程序空间的数据	R0 ← (Z)	None	3
LPM	Rd, Z	加载程序空间的数据	Rd ← (Z)	None	3
LPM	Rd, Z+	加载程序空间的数据，然后地址加一	Rd ← (Z), Z ← Z+1	None	3
SPM		保存程序空间的数据	(Z) ← R1:R0	None	-
IN	Rd, P	从 I/O 端口读数据	Rd ← P	None	1

指令	操作数	说明	操作	标志	# 时钟数
OUT	P, Rr	想 I/O 端口输出数据	P ← Rr	None	1
PUSH	Rr	将寄存器推入堆栈	STACK ← Rr	None	2
POP	Rd	将寄存器弹出堆栈	Rd ← STACK	None	2
MCU 控制指令					
NOP		空操作		None	1
SLEEP		休眠	(see specific descr. for Sleep function)	None	1
WDR		复位看门狗	(see specific descr. for WDR/timer)	None	1
BREAK		终止	For On-chip Debug Only	None	N/A

Note: 1. 这些指令只对 ATmega168 有效

产品信息

ATmega48

工作速度 (MHz)	电 源	定 货 号	封 装	工作温度范围
2[3]	1.8 - 5.5	ATmega48V-2AI ATmega48V-2PI ATmega48V-2MI ATmega48V-2AJ[2] ATmega48V-2PJ[2] ATmega48V-2MJ[2]	32A 28P3 32M1-A 32A 28P3 32M1-A	工业级 (-40°C - 85°C)
16[3]	2.7 - 5.5	ATmega48-16AI ATmega48-16PI ATmega48-16MI ATmega48-16AJ[2] ATmega48-16PJ[2] ATmega48-16MJ[2]	32A 28P3 32M1-A 32A 28P3 32M1-A	工业级 (-40°C - 85°C)

Note: 1. 产品也可以 wafer 的形式提供，订货信息细节以及最小定货量请与 Atmel 当地机构联系。
2. 可选无铅封装
3. 参见 P276Figure 131 与 P277Figure 132 。

封装类型	
32A	32- 引线，薄 (1.0 mm) 塑料鸥翼方形扁平式封装 (TQFP)
28P3	28- 引线， 0.300” 宽， Plastic Dual Inline Package (PDIP)
32M1-A	32- 焊垫， 5 x 5 x 1.0mm 大小，线距 0.50 mm，微导线框封装 (MLF)

ATmega88

工作速度 (MHz)	电 源	定 货 号	封 装	工作温度范围
2[3]	1.8 - 5.5	ATmega88V-2AI ATmega88V-2PI ATmega88V-2MI ATmega88V-2AJ[2] ATmega88V-2PJ[2] ATmega88V-2MJ[2]	32A 28P3 32M1-A 32A 28P3 32M1-A	工业级 (-40°C - 85°C)
16[3]	2.7 - 5.5	ATmega88-16AI ATmega88-16PI ATmega88-16MI ATmega88-16AJ[2] ATmega88-16PJ[2] ATmega88-16MJ[2]	32A 28P3 32M1-A 32A 28P3 32M1-A	工业级 (-40°C - 85°C)

Note: 1. 产品也可以 wafer 的形式提供，订货信息细节以及最小定货量请与 Atmel 当地机构联系。
2. 可选无铅封装
3. 参见 P276Figure 131 与 P277Figure 132 。

封装类型	
32A	32- 引线，薄 (1.0 mm) 塑料鸥翼方形扁平式封装 (TQFP)
28P3	28- 引线， 0.300” 宽， Plastic Dual Inline Package (PDIP)
32M1-A	32- 焊垫， 5 x 5 x 1.0mm 大小，线距 0.50 mm，微导线框封装 (MLF)

ATmega168

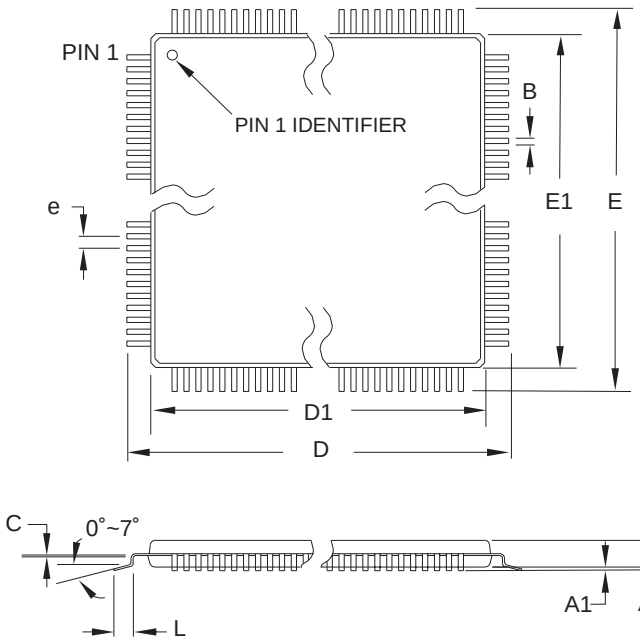
工作速度 (MHz)	电 源	定 货 号	封 装	工作温度范围
2[3]	1.8 - 5.5	ATmega168V-2AI ATmega168V-2PI ATmega168V-2MI ATmega168V-2AJ[2] ATmega168V-2PJ[2] ATmega168V-2MJ[2]	32A 28P3 32M1-A 32A 28P3 32M1-A	工业级 (-40°C - 85°C)
16[3]	2.7 - 5.5	ATmega168-16AI ATmega168-16PI ATmega168-16MI ATmega168-16AJ[2] ATmega168-16PJ[2] ATmega168-16MJ[2]	32A 28P3 32M1-A 32A 28P3 32M1-A	工业级 (-40°C - 85°C)

Note: 1. 产品也可以 wafer 的形式提供，订货信息细节以及最小定货量请与 Atmel 当地机构联系。
2. 可选无铅封装
3. 参见 P276Figure 131 与 P277Figure 132 。

封装类型	
32A	32- 引线，薄 (1.0 mm) 塑料鸥翼方形扁平式封装 (TQFP)
28P3	28- 引线， 0.300” 宽， Plastic Dual Inline Package (PDIP)
32M1-A	32- 焊垫， 5 x 5 x 1.0mm 大小，线距 0.50 mm，微导线框封装 (MLF)

封装信息

32A



COMMON DIMENSIONS
(Unit of Measure = mm)

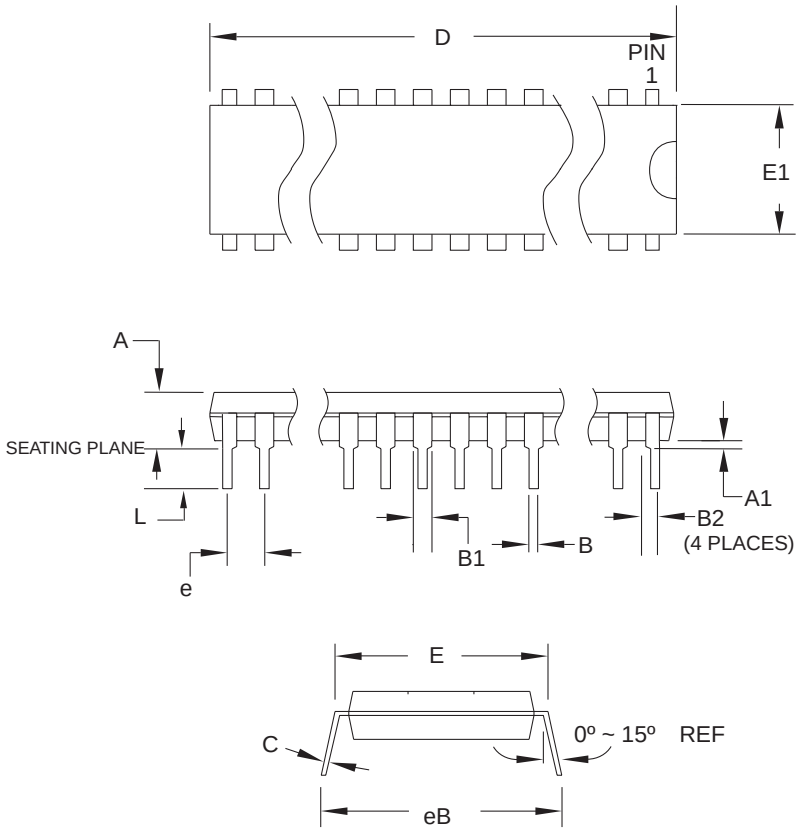
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation ABA.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

10/5/2001

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**32A,** 32-lead, 7 x 7 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)	32A	B

28P3



COMMON DIMENSIONS
(Unit of Measure = mm)

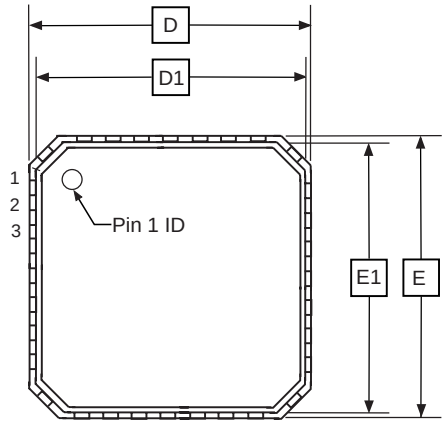
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.5724	
A1	0.508	–	–	
D	34.544	–	34.798	Note 1
E	7.620	–	8.255	
E1	7.112	–	7.493	Note 1
B	0.381	–	0.533	
B1	1.143	–	1.397	
B2	0.762	–	1.143	
L	3.175	–	3.429	
C	0.203	–	0.356	
eB	–	–	10.160	
e	2.540 TYP			

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

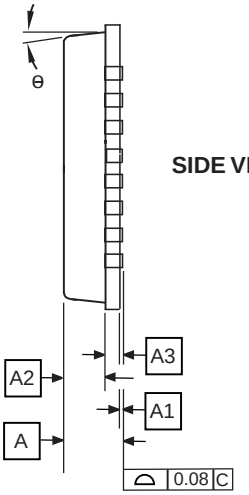
09/28/01

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**28P3**, 28-lead (0.300"/7.62 mm Wide) Plastic Dual Inline Package (PDIP)	28P3	B

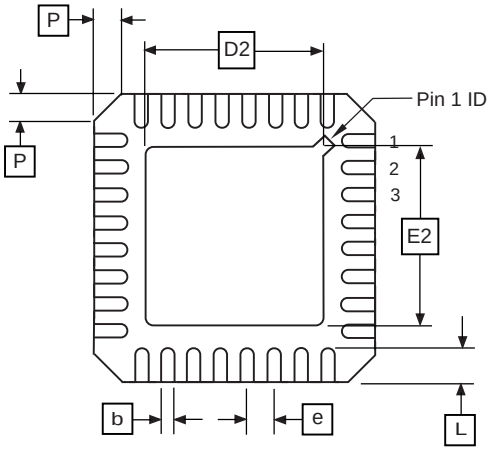
32M1-A



TOP VIEW



SIDE VIEW



BOTTOM VIEW

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A2	–	0.65	1.00	
A3		0.20 REF		
b	0.18	0.23	0.30	
D		5.00 BSC		
D1		4.75 BSC		
D2	2.95	3.10	3.25	
E		5.00 BSC		
E1		4.75BSC		
E2	2.95	3.10	3.25	
e		0.50 BSC		
L	0.30	0.40	0.50	
P	–	–	0.60	
θ	–	–	12°	

Notes: 1. JEDEC Standard MO-220, Fig. 2 (Anvil Singulation), VHHD-2.

01/15/03

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**32M1-A**, 32-pad, 5 x 5 x 1.0 mm Body, Lead Pitch 0.50 mm Micro Lead Frame Package (MLF)	32M1-A	C

ATmega48 勘误表

本节的版本号与 ATmega48 器件的版本号相同。

Rev A

- **在擦除操作后读返回值错误**

1. 在擦除操作后读返回值错误

当电源电压低于 2.7 V 时，被擦除的 EEPROM 地址读返回值为 0x00，而不是 0xFF。

解决方法：

如果要在擦除操作之后进行读操作，请使用原子写操作，数据为 0xFF。这同样可以达到擦除的目的。在任何情况下写操作都是正确的。

ATmega88 勘误表

本节的版本号与 ATmega88 器件的版本号相同。

Rev A

- **在擦除操作后读返回值错误**

1. 在擦除操作后读返回值错误

当电源电压低于 2.7 V 时，被擦除的 EEPROM 地址读返回值为 0x00，而不是 0xFF。

解决方法：

如果要在擦除操作之后进行读操作，请使用原子写操作，数据为 0xFF。这同样可以达到擦除的目的。在任何情况下写操作都是正确的。

ATmega168 勘误表

本节的版本号与 ATmega168 器件的版本号相同。

Rev A

- **在擦除操作后读返回值错误**

1. 在擦除操作后读返回值错误

当电源电压低于 2.7 V 时，被擦除的 EEPROM 地址读返回值为 0x00，而不是 0xFF。

解决方法：

如果要在擦除操作之后进行读操作，请使用原子写操作，数据为 0xFF。这同样可以达到擦除的目的。在任何情况下写操作都是正确的。

内容表

特性 1

引脚排列 2

总览 3

方框图 3
ATmega48、 ATmega88 和 ATmega168 之间的比较 4
Disclaimer 5
引脚说明 5

代码例子 6

AVR CPU 内核 7

综述 7
结构总览 7
ALU – 算逻单元 8
状态寄存器 9
通用寄存器文件 10
堆栈指针 11
指令执行时序 12
复位和中断处理 12

AVR ATmega48/88/168 的存储器 15

在系统内可编程的 Flash 程序存储器 15
SRAM 数据存储器 17
EEPROM 数据存储器 19
I/O 存储器 25

系统时钟和时钟选项 26

时钟系统及其分布 26
时钟源 27
低功耗晶体振荡器 29
满振幅晶体振荡器 30
低频晶体振荡器 32
标定的片内 RC 振荡器 32
128 kHz 片内振荡器 33
外部时钟 34
时钟输出缓冲器 35
定时器 / 计数器振荡器 35
系统时钟预分频器 35

电源管理和睡眠模式 38

空闲模式 39
ADC 噪声抑制模式 39
掉电模式 39
省电模式 39

Standby 模式 40
最小化芯片功耗..... 40

系统控制和复位..... 42
片内电压基准 48
看门狗定时器 49
通过有序命令改变看门狗定时器的配置..... 53

中断..... 54
ATmega48 的中断向量 55
ATmega88 的中断向量 57
ATmega168 的中断向量 60

I/O 端口..... 65
综述..... 65
作为通用数字 I/O 的端口 66
端口的第二功能..... 70
I/O 端口的寄存器说明 82

外部中断 83

具有 PWM 功能的 8 位定时器 / 计数器 0 88
综述..... 88
定时器 / 计数器的时钟源 89
计数器单元..... 89
输出比较单元 90
比较匹配输出单元..... 92
工作模式 93
定时器 / 计数器时序框图 97
8 位定时器 / 计数器寄存器的说明 99

定时器 / 计数器 0 和 定时器 / 计数器 1 的预分频器..... 105

16 位定时器 / 计数器 1 107
综述..... 107
访问 16 位的寄存器..... 110
定时器 / 计数器的时钟源 113
计数器单元..... 113
输入捕捉单元 114
输出比较单元 116
比较匹配输出单元..... 118
工作模式 119
定时器 / 计数器时序框图 127
16 位定时器 / 计数器寄存器的说明 129

8-bit Timer/Counter2 with PWM and Asynchronous Operation .. 136

综述 136
时器 / 计数器的时钟源 137
计数器单元 137
输出比较单元 138
比较匹配输出单元 140
工作模式 141
定时器 / 计数器时序框图 145
8 位定时器 / 计数器寄存器的说明 147
异步工作的定时器 / 计数器 152
定时器 / 计数器的预分频器 156

串行外设接口 – SPI 157
$\overline{SS}$ 引脚的功能 162
数据模式 165

USART0 166
综述 166
时钟的产生 167
数据帧格式 170
USART 初始化 171
数据发送 – USART 发送器 173
数据接收 – USART 接收器 176
异步数据接收 179
多处理器通讯模式 183
USART 寄存器的说明 184
波特率设置的例子 189

工作于 SPI 模式的 USART 193
综述 193
时钟的产生 193
SPI 数据模式和时序 194
数据帧格式 194
数据发送器 196
USART MSPIM 寄存器的说明 199
AVR USART MSPIM 及
AVR SPI 201

2 线串行接口 203
特点 203
2 线蹕行接口总线定义 203
数据发送器和数据帧格式 204
多主机总线系统，仲裁和同步 207
TWI 模块综述 209
TWI 寄存器的说明 211
使用 TWI 215
数据发送模式 218

多主机系统和仲裁 231

模拟比较器 233
模拟比较器的多工输入 235

AD 转换器 236
特点 236
开始转换 238
预分频器和转换时序 239
改变通道或基准 242
ADC 噪声抑制 243
ADC 转换结果 247

debugWIRE：片上调试系统 252
特点 252
综述 252
物理接口 252
Software Break Points 软件断点 253
debugWIRE 的限制 253
debugWIRE 相关的寄存器 253

ATmega48 Flash 的自编程 254
自编程时 Flash 的寻址 255

Boot Loader 支持 – Read-While-Write 自编程 , ATmega88 和 ATmega168 261
Boot Loader 特点 261
应用区和 Boot Loader 区 261
Read-While-Write 和 非 Read-While-Write Flash 区 261
Boot Loader 锁定位 264
Boot Loader 编程 265
自编程时 Flash 的寻址 267
Flash 的自编程 268

存储器的编程 276
程序和数据存储器锁定位 276
熔丝位 278
标识位 280
标定字节 280
并行编程参数，引脚映射和命令 281
串行编程引脚映 283
P 并行编程 284
串行下载 292

电气特性 296
最大值 *..... 296
DC 特性 296
外部时钟驱动波形 298
外部时钟驱动 298
2 线串行接口特性 300
SPI 时序特性 301
ADC 特性 – 初始数据 303

ATmega48/88/168 典型数据 – 初始数据 304

寄存器摘要 332

指令集摘要 336

定货信息 339
ATmega48 339
ATmega88 340
ATmega168 341

封装信息 342
32A 342
28P3 343
32M1-A 344

ATmega48 勘误 345
Rev A 345

ATmega88 勘误 346
Rev A 346

ATmega168 勘误 347
Rev A 347

内容表 i

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe

Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia

Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan

9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/ High Speed Converters/RF Datacom

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests

www.atmel.com/literature

Disclaimer: Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

© Atmel Corporation 2003. **All rights reserved.** Atmel® and combinations thereof, AVR®, and AVR Studio® are the registered trademarks of Atmel Corporation or its subsidiaries. Microsoft®, Windows®, Windows NT®, and Windows XP® are the registered trademarks of Microsoft Corporation. Other terms and product names may be the trademarks of others

Printed on recycled paper.

2545A–AVR–09/03